VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Nástroje pro zátěžové testy

Tools for load testing

2011/2012 Lukáš Navrátil

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Navrátil**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Nástroje pro zátěžové testy
Tools for Load Testing

Zásady pro vypracování:

Cílem závěrečné práce je vytvořit souhrnný přehled komerčních a open source nástrojů pro zátěžové testování webových aplikací, tyto nástroje porovnat a na konkrétních příkladech popsat jejich výhody a nevýhody.
1. Popsat výhody a nevýhody komerčních a open source nástrojů pro zátěžové testování na konkrétních příkladech.
2. Pro vybranou webovou aplikaci vytvořit zátěžový test a vyhodnotit výsledky.

Seznam doporučené odborné literatury:

PATTON, R.: Testování softwaru
KROLL, P.: Rational Unified Process Made Easy, The: A Practitioner's Guide to the RUP
MOLYNEAUX, I.: The Art of Application Performance Testing: Help for Programmers and Quality Assurance

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Roman Šebesta, Ph.D.**

Datum zadání: 19.11.2010
Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

„Prohlašuji , že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal."

V Ostravě dne: 4.5.2012

Podpis: .....

Poděkování

Rád bych na tomto místě poděkoval vedoucímu své bakalářské práce, panu Ing. Romanovi Šebestovi, Ph.D., za ochotu, cenné rady a připomínky při vedení mé bakalářské práce.

Dále bych pak chtěl poděkovat, panu Miloši Kalhousovi, z firmy T-mobile, který mi byl nápomocen při řešení první části mé bakalářské práce.

A také bych chtěl poděkovat konzultantovi pro druhou část této bakalářské práce, kterým byl pan Petr Czaderna, za jeho rady, náměty a připomínky při psaní této práce.

Abstrakt

Tato bakalářská práce řeší problematiku zátěžových testů. Nejprve je uvedena teoretická část popisované problematiky. Testování je zasazeno do rámce softwarového procesu. Jsou uvedeny jednotlivé typy testování, s důrazem na zátěžové testy. Tato práce má dva cíle. Jako první se zabývá zátěžovým testováním konkrétní webové aplikace. Tuto webovou aplikaci poskytla firma T-Mobile. Pro tuto aplikaci bylo provedeno několik testů, a to ve dvou vybraných nástrojích, Apache JMeter a The Grinder. V druhé části jsou popsány vybrané zátěžové testovací nástroje, které se objevují na trhu. Jsou vybráni čtyři představitelé komerčních nástrojů a to sice: HP LoadRunner, IBM Rational performance tester, SilkPerformer a Wapt. Z open source nástrojů jsou vybrány tři: Apache JMeter, Clif a The Grinder. Na závěr bakalářské práce jsou tyto nástroje pro zátěžové testy porovnány z několika hledisek.

Klíčová slova

Softwarový proces, Zátěžové testy, Testovací nástroje, Testování webových aplikací, Open source testovací nástroje, HP LoadRunner, IBM Rational performance tester, SilkPerformer, Wapt, Apache JMeter, Clif, The Grinder

Abstract

This bachelor thesis solves the problem of load testing. At first is given the theoretical part of discussed issue. Testing is set in the framework of software process. They are given various types of testing with emphasis on load testing. This thesis has two aims. First deals with specific load testing of concrete web application. This web application is provided by T-Mobile company. For this application were done several tests in two selected tools: Apache JMeter and The Grinder. The second section describes selected load testing tools that appear on the market. They are chosen four representatives of commercial tools: HP LoadRunner, IBM Rational performance tester, SilkPerformer and Wapt. Out of open source tools are chosen three: Apache JMeter, Clif and The Grinder. In conclusion of this bachelor thesis are these tools for load testing compared from several aspects.

Key words

Software process, Load testing, Testing tools, Testing of web applications, Open source testing tools, HP LoadRunner, IBM Rational performance tester, SilkPerformer, Wapt, Apache JMeter, Clif, The Grinder

Seznam použitých symbolů a zkratek

.NET – NETwork, soubor technologií
AJAX – Asynchronous JavaScript and XML, technologie pro vývoj webových aplikací
ASP.NET – Active Server Pages .NET, technologie
API – Application Programming Interface, rozhraní pro programování aplikací
BDL – Benchmark Description Language, testovací skriptovací jazyk
BSD – Berkeley Software Distribution, druh licencí
CPU – Central Processing Unit, procesor
FAQ – Frequently Asked Questions, často kladené otázky
FTP – File Transfer Protocol, protokol pro přenos souborů
GUI – Graphical User Interface, grafické uživatelské rozhraní
HP – Hewlett-Packard, mezinárodní společnost
HTML – HyperText Markup Language, značkovací jazyk
HTTP – Hypertext Transfer Protocol, internetový protokol
HTTPS – Hypertext Transfer Protocol Secure, internetový protokol
HW – Hardware
IBM – International Business Machines Corporation, mezinárodní společnost
JDK – Java Development Kit, soubor nástrojů pro aplikaci Java
JSON – JavaScript Object Notation, standard
JVM – Java Virtual Machine, sada počítačových programů
LDAP – Lightweight Directory Access Protocol, protokol pro přístup k datům
ODBC – Open Database Connectivity, standardizované softwarové API
RAM – Random-Access Memory, druh počítačové paměti
REST – Representational state transfer, softwarová architektura
RDP – Remote Desktop Protocol, síťový protokol
SAP – název softwarových řešení od firmy SAP (System Analysis and Program Development)
SIP – Session Initiation Protocol, internetový protokol
SOA – Service-oriented architecture, soubor zásad a metodik
SOAP – Simple Object Access Protocol, protokol
SQL – Structured Query Language, dotazovací jazyk
SP3 – Service Pack 3, balíček aktualizací
TCP – Transmission Control Protocol, internetový protokol
TCP/IP – Transmission Control Protocol/Internet Protocol, sada protokolů
XHTML – Extensible HyperText Markup Language, značkovací jazyk
XML – Extensible Markup Language, značkovací jazyk
XPath – XML Path Language, počítačový jazyk

Obsah

1 Úvod

V dnešní době již internet není jen výsadou odborníků, ale má k němu přístup široká veřejnost. Takže se z internetu, místa, kde se sdílejí informace, stalo působiště celé řady firem, jež se snaží prosadit. Pro většinu internetových aplikací je důležitým ukazatelem jejich návštěvnost. Ta se dá podpořit různými typy reklamy. To je ovšem jen část klíče k úspěchu, hlavní snahou je zajistit celkovou kvalitu webových aplikací a jejich bezproblémový chod. Očekávaná doba odezvy aplikací se v poslední době, také velmi zkrátila. Webové aplikace tak potřebují mít zajištěnu jakost i v tomto směru. Jak ale ověřit, že se webová aplikace bude chovat tak, jak požadujeme i při nasazení do provozu? Odpověď nám poskytuje zátěžové testování.

V úvodu práce jsem popsal obecně průběh vývoje softwaru a následně jsem uvedl, jak s ním souvisí testovaní a také, jaké důvody k němu vedou. Dále jsem přiblížil problematiku testování, postupy využívané při testování, jednotlivé druhy testování a druhy testovacích nástrojů, které se běžně využívají. V další části jsem se pak podrobněji věnoval problematice zátěžových testů, jeho rozdělení a sledovaným faktorům.

Následuje popis způsobu práce s jednotlivými porovnávanými nástroji, který vznikl na základě mých vlastních poznatků při práci s programy. Praktická část bakalářské práce popisuje průběh testování firemní aplikace od návrhu testu až po jeho provedení a analýzu výsledků.

V kapitole „Chyba: zdroj odkazu nenalezen“ jsou pak porovnány jednotlivé nástroje podle konkrétních stanovených kritérií.

2 Vývoj softwarového projektu

2.1 Proces vývoje softwaru

Proces vývoje softwaru zahrnuje celou řadu činností a metod, které vedou k vytvoření softwarového produktu. Jako základní kámen je stanoven tzv. softwarový proces, což je posloupnost činností, potřebných k vytvoření daného softwarového produktu. Jak říká definice: „Softwarový proces je po částech uspořádaná množina kroků, směřujících k vytvoření nebo úpravě softwarového díla.“ [1]

Každý softwarový proces lze rozdělit na jednotlivé fáze: zahájení, rozpracování, tvorbu a předání. Fáze zahájení, kromě jiných, obsahuje první náčrt projektu, zjištění možných rizik a sepsání produktové dokumentace. Při rozpracování projektu se řeší konkrétní detaily specifikace a je rozpracována konečná architektura produktu. V části tvorby pak dochází k implementaci v konkrétním programovacím jazyce a sestavení softwaru do spustitelné podoby. Ve fázi poslední je produkt předán k užívání koncovému uživateli.

2.2 Modely vývoje softwaru

Existuje celá řada modelů softwarového procesu. Mezi ty nejznámější a nejpoužívanější patří vodopádový model, prototypový přístup, inkrementální přístup, spirálový model a další. Pro nás je především podstatné, že ve všech těchto modelech či postupech má, nebo by mělo mít, své místo testování softwaru.

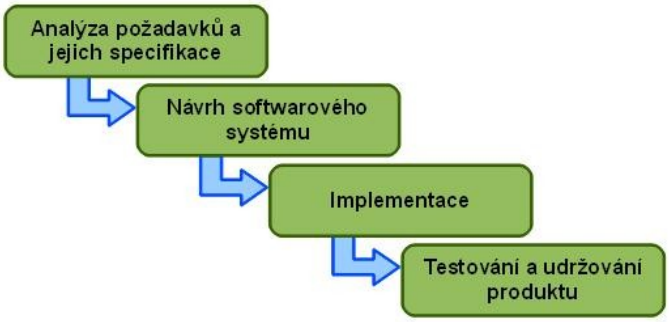
2.2.1 Vodopádový model

Vodopádový model je rozdělen do čtyř hlavních částí:

- analýza požadavků a jejich specifikace,
- návrh softwarového systému,
- implementace,
- testování a udržování produktu.

Způsob fungovaní tohoto postupu je v podstatě jednoduchý. Každá část se provádí odděleně a po jejím ukončení se začne provádět část další. „Jinými slovy řečeno, výsledky předchozí fáze „vtékají“ jako vstupy do fáze následující.“[1] Odtud tedy název „vodopádový model“. Díky jednoduchosti se stal základem pro mnoho dalších přístupů, které se snaží především odbourat nedostatky tohoto modelu, mezi něž patří:

- skutečnost, že po dokončení prací na jedné z etap, se k ní již není možno vrátit, a tak úspěšnost projektu hodně závisí na kvalitním provedení jednotlivých částí,
- jak bude výsledný software vypadat, je možno vidět až v poslední fázi vývoje, kdy už je pozdě k vykonání větších změn v projektu,
- změny v zadání projektu budou většinou znamenat spuštění vývoje od začátku.



Obrázek 1: Vodopádový model

2.2.2 Inkrementální přístup

Vychází z vodopádového modelu. Při vývoji se postupuje tak, že se celý projekt rozdělí na určitý počet částí a v každé části se postupuje, jako by to byl samostatný projekt s vodopádovým modelem. Po ukončení jedné části se proces opakuje a produkt se takto postupně rozrůstá do výsledné podoby. Výhodou oproti předchozímu postupu je, že po ukončení každého cyklu vývoje vidíme funkční prototyp softwaru a chyby, které se objeví v jednotlivých částech, pak můžeme v další iteraci vývoje opravit.

2.2.3 Spirálový model

„Spirálový model náleží do skupiny tzv. přístupů řízených riziky. To znamená, že postup do další fáze závisí na důsledně provedené analýze všech rizik a možných problémů.“ [2] Je svým způsobem podobný inkrementálnímu přístupu, přičemž klade důraz na důkladnou analýzu mezi jednotlivými iteracemi vývoje. Vychází se z požadavků, které jsou na počátku pouze hrubým náčrtem výsledné aplikace a postupně se upřesňují podle toho, jak vývoj produktu postupuje.

„Celý životní cyklus podle Spirálového modelu je rozdělen do čtyř hlavních částí:

- určení cílů, alternativ a omezení,

- vyhodnocení alternativ, identifikace a řešení rizik,
- vývoj a verifikace další úrovně produktu,
- plánování následujících fází.

Po každé fázi následuje testování, hodnocení a předání dílčích výsledků.“ [2]

2.2.4 Rational Unified Process (RUP)

Jedná se o profesionální „produkt“, který vyvíjí společnost IBM (International Business Machines Corporation), dříve jej vyvíjela společnost Rational (odtud také název). Obsahuje efektivní postupy osvědčené při vývoji, či zavádění softwaru. Jejich použití zamezí vynalézání některých částí, které již byly dříve vymyšleny. Tyto části se jako komponenty použijí z knihovny „Rational Process Library“ a poté se pouze upraví podle potřeb daného projektu. Tento model vývoje má několik postupů, které jsou doporučeny dodržovat:

- „Softwarový produkt je vyvíjen iteračním způsobem.
- Jsou spravovány požadavky na něj kladené.
- Využívá se již existujících softwarových komponent.
- Model softwarového systému je vizualizován.
- Průběžně je ověřována kvalita produktu.
- Změny systému jsou řízeny.“ [1]

2.3 Testování v rámci projektu vývoje

Uvedli jsme si typy modelů, podle kterých probíhá vytváření softwaru, jejich specifika a fáze průběhu. Nyní se zaměříme na jednotlivé etapy, popíšeme jejich spojitost s testováním a vysvětlíme, kdy se hodí začít s jednotlivými typy testů. Testovat můžeme v podstatě ve všech fázích vývoje, a to vždy ve chvíli, kdy máme hotovou některou konkrétní část, či celek.

Jak už bylo řečeno, jako první při tvorbě každého projektu, by se mělo začít od specifikace. A již po vytvoření specifikace, či její části, se můžeme pustit do jejího otestování. Celkově bychom se z pohledu testera měli snažit ve specifikaci najít všechny části, které by mohly v pozdějších fázích dělat problémy.

Po specifikaci nastává čas na návrh. Při návrhu se rámcově promýšlí struktury nově vznikajícího softwaru. Tato část je z pohledu bezchybnosti velice důležitá, protože problémy, které se v této části neodhalí, se pak vlečou s projektem, až k jeho dokončení. A pochopitelně čím dříve případnou chybu odhalíme, tím menší náklady na její odstranění budeme potřebovat. Tester, který se zabývá touto problematikou, by měl mít znalost struktur, ve kterých se provádí návrh produktu. Pochopitelně by měl mít i podrobnou znalost technologie, ve které se bude později projekt vytvářet. Jeho cílem je totiž ověřit, jestli návrhy bude možno korektně realizovat a jestli vše odpovídá a bude odpovídat produktové specifikaci. Pokud máme k dispozici návrhy uživatelského rozhraní, můžeme provést první testy použitelnosti, jež sledují především faktory snadného a přívětivého užívání budoucí aplikace.

Dále následuje část vytváření zdrojového kódu programu a s ní souvisí část testování, kterou si při výrazu „testování softwaru“ asi nejčastěji vybavíme. Je to testování zdrojového kódu z pohledu bezchybnosti. Testování může probíhat jak na kódu, který se spouští, to jsou tzv. dynamické testy, které zahrnují testování různých funkcí a akcí v aplikaci, tak i na kódu, který ještě nemá spustitelnou podobu, potom provádíme statické testování, které zahrnuje například revize kódu.

V závěrečných fázích vývoje, nebo iterace pak provádíme testy, které potřebují pro své fungování minimálně spustitelný prototyp programu. Mezi ně patří například testy konfigurace

a testy kompatibility, které ověřují bezproblémový běh vytvořeného softwaru pro konkrétní hardwarové a softwarové konfigurace. Na uživatelském rozhraní provádíme další testování použitelnosti. Také můžeme provádět různé druhy zátěžových testů, čímž ověříme, jestli aplikace odpovídá požadavkům, které na ni budou kladeny při skutečném provozu.

Není nutné, zvláště pokud vyvíjíme iterativním způsobem, pokaždé provádět všechny typy testování. Situace se mění od projektu. Některé druhy testů můžeme provádět v pravidelných intervalech a jiné až po skončení určité etapy.

V této podkapitole jsou uvedeny pojmy, které se pokusím v dalších částech postupně vysvětlit. Jedná se především o objasnění toho, co si máme představit jako chybu (podkapitola 2.5), dále jsou uvedeny jednotlivé typy testování, kterým se budu věnovat v kapitole 3.3.

2.4 Testování softwaru

Podstatou testování je nalézt chyby, které obsahuje testovaný software. Chybou ovšem nemusí být jen chybový stav aplikace. Tomu, co je chyba se bude věnovat následující podkapitola. Výstupem z testování by měla být dokumentace, která zachycuje průběh testování a přehledně popisuje nalezené chyby. Důležité jsou pak především informace, nakolik je chyba závažná, jaké chování bylo očekáváno, jak se program při chybě zachoval, ale především jak k chybě došlo. Na základě této dokumentace, by měl být schopen programátor odhalit, čím bylo chování aplikace způsobeno.

U řady projektů se stává, že z finančních důvodů není testování věnována patřičná pozornost, nebo vzhledem k tomu, že se testuje až ke konci každé iterace vývoje, na něj prostě nezbude dostatečný čas. A tak do další fáze vývoje, nebo dokonce přímo k zákazníkovi, putuje software, který není řádně otestován a není tudíž zajištěna jeho jakost.

2.5 Co je to chyba

Jak již bylo v textu uvedeno, ke každému softwaru by měla vznikat také jeho dokumentace. A právě specifikace produktu je klíčem k odhalení chyb. Ovšem i samotná specifikace může obsahovat chybu, a proto je nutné provádět testování od prvních kroků vývoje softwarového produktu. Rozlišit můžeme mnoho druhů chyb, záleží především na tom, který druh testování právě provádíme a podle toho se určuje i škála hledaných závad. Obecně jde jako chybu označit cokoliv, co bude snižovat hodnotu výsledného produktu. Podle definice pak za chybu považujeme, když je splněna některá z těchto podmínek:

- „Software nedělá něco, co by podle specifikace produktu dělat měl.
- Software dělá něco, co by podle údajů specifikace dělat neměl.
- Software dělá něco, o čem se produktová specifikace nezmiňuje.
- Software nedělá něco, o čem se produktová specifikace nezmiňuje, ale měla by se zmiňovat.
- Software je obtížně srozumitelný, těžko se s ním pracuje, je pomalý – nebo podle názoru testera softwaru – jej koncový uživatel nebude považovat za správný.“ [3]

Jak je vidět, chyby lze rozdělit do dvou kategorií. A to sice na chyby, které jsou přesně definované. A na chyby, jejichž posouzení závisí na zvážení testera. Především u druhé kategorie chyb se projeví zkušenosti a schopnosti každého testera.

2.6 Proč testovat

Nabízí se otázka „Proč vlastně software testovat?“. Testování jako takové funkčnost softwarového produktu již dále nerozvíjí. V neotestované aplikaci se však mohou, a nejspíše i budou, nacházet chyby. Přínos testování tedy spočívá v tom, že můžeme bez větších pochybností tvrdit, že software funguje tak, jak má. Zvyšuje se tak velmi podstatně pravděpodobnost, že zákazník bude s výsledným produktem spokojen. V ideálním případě se samotného testování také sám účastní, protože je to zákazník, kdo může nejlépe posoudit, co od softwaru očekává. Je praxí prokázané, že ačkoliv se na testování musí vyhradit čas a prostředky, dojde ve výsledku k ušetření času, prostředků a tím i nákladů. Přehlédnutá chyba, nebo chyby, mohou být natolik závažné, že výsledný produkt se špatně používá, nebo je nepoužitelný úplně. Také je zde pravidlo, které říká, že čím dříve je chyba v softwarovém produktu zjištěna a je z něj odstraněna, tím menší jsou potom náklady na její pozdější likvidaci.

Jak již bylo řečeno, při vývoji softwarového díla musí jako první vzniknout specifikace, podle které se bude dále postupovat. Specifikaci vytváří buď zákazník sám anebo ještě lépe zákazník společně s vývojovým týmem. Práce testera pak spočívá v tom, že dostane produktovou specifikaci a na základě ní hledá chyby ve výsledném produktu, nebo jeho části. Důležité je, přesně specifikovat, co vlastně můžeme považovat za chybu a co ne. Software je možno vytvářet i bez toho, že by se vycházelo z průvodní dokumentace, ovšem proces testování je založen na tom, že víme, jak má výsledný produkt vypadat. Pokud to nevíme, je již pouze na zvážení testera co a jak bude testovat a považovat za správné. Nicméně bez dokumentace se dají efektivně vytvářet jen velmi malé projekty, kterými se v tomto textu nebudeme zabývat.

Zdroje pro kapitolu 2: [1], [2], [3].

3 Problematika testování

3.1 Přístupy k testování

3.1.1 Black box testing

Black box testing, neboli testování černé skřínky. Tento přístup k testování je specifický v tom, že tester nezná žádné implementační detaily softwarového produktu. Pracuje se softwarem podobně jako by byl koncový zákazník. Při takovémto testování vychází z produktové specifikace. Sestavuje z ní množinu testovacích hodnot a také předpokládané výstupy. Vzhledem k tomu, že neznáme vnitřní strukturu softwaru, je třeba počítat s možností pádu systému. Naopak chyby objevené při white box testingu se dají touto metodou ověřit a demonstrovat.

Výhody:

- Metoda je velice rychlá. Umožňuje v krátké době ověřit velké množství vstupů.
- Je dobře srozumitelná nejen odborníkům, ale i zákazníkům. K psaní scénářů není třeba umět programovat a znát konkrétní technologie, popřípadě si vystačíme se základními znalostmi.
- Implementace na test nemá vliv, proto je možno testovací scénáře napsat už ve chvíli, kdy je hotová specifikace a nemění se ani při změnách v implementaci.
- Utajení. Tester, nebo testovací tým nemusí znát zdrojový kód.

Nevýhody:

- Přestože software generuje správné výsledky, může provádět další akce, které jsou v rozporu se zamýšleným chováním.

- Neodhalí neefektivní implementace.

3.1.2 White box testing

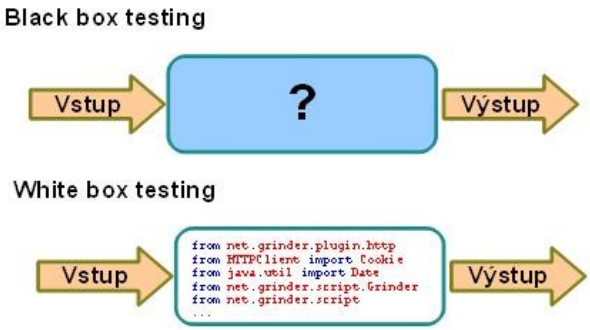
White box testing, neboli testování bílé skříňky. U tohoto způsobu testování máme přesný přehled o tom, co se děje uvnitř programu. Známe programový kód i technologii, ve které je projekt vyvíjen a v některých případech také použitý hardware. Tester analyzuje zdrojový kód a poté provádí jeho revizi. Musí při tom na kód nahlížet pokud možno nezaujatě. To znamená, že by neměl předpokládat, že něco funguje, jak má. Měl by zjistit, jak programový kód skutečně pracuje a pak ověřit, jestli je to tak správně. Další důležitou částí revizí je vyhledávání nevyužitých a zbytečných částí kódu, které vlastně nic nedělají a mohou být potenciálně nebezpečné. Vhodné je však jít ještě dále a identifikovat ty části kódu, které nejsou efektivní. Jejich přepracování pak dále může zrychlit a zkvalitnit vyvíjený produkt. Správně provedené white box testování, by tedy mělo programový kód dovést do stavu, kdy je maximálně efektivní a využívá minimum systémových prostředků. Samozřejmě i zde musíme vzít v úvahu finanční a časové omezení a najít vhodný kompromis mezi kvalitou a náklady. Protože drahý a dlouze vyvíjený produkt není nejlepším řešením.

Výhody

- Je schopen odhalit chyby dříve, než jsou implementovány. Šetří tak čas i prostředky, které by bylo třeba vyhradit na pozdější nápravu chyb.
- Odhalení nežádoucích částí kódu a neefektivní implementace.

Nevýhody

- Tester musí detailně znát technologii, ve které je napsán produkt.
- Je potřeba mít zkušené testery a sofistikovaný testovací software, což znamená zvýšené náklady.



Obrázek 2: Ilustrace rozdílu mezi testováním černé a bílé skříňky

3.1.3 Grey box testing

Grey box testing, neboli testování šedé skříňky. Je kombinace předcházejících dvou technik. Zahrnuje omezenou znalost vnitřních datových struktur programu. Tester je díky tomu schopen přesněji navrhnout testovací scénáře, které teď již nemusí odhadovat, ale může je přesně stanovit na základě znalosti kódu. Samotné testování probíhá jako u černé skříňky.

Výhody

- Slučuje některé výhody black a white box testování.

- Tester díky částečné znalosti vnitřní struktury ušetří čas, který by jinak musel investovat do zkoumání programu.

Nevýhody

- Tester opět nemá k dispozici celý zdrojový kód testovaného produktu, takže neefektivní implementace a podobné příznaky nezdravého kódu, zůstanou skryty.

3.2 Testovací postupy

Samotné testování s sebou nese mnoho různých postupů a technik, jak dosáhnout s co největší pravděpodobností bezchybného softwaru. Softwarový tester pracuje podobně jako běžný uživatel, s tím zásadním rozdílem, že se snaží s aplikací pracovat i způsobem, který by se nepředpokládal. Snaží se opakovat různé akce s různými hodnotami.

Tento záměr však skýtá zásadní problém. Každý, byť jednoduchý program má spoustu možností, jak s ním pracovat. Často je to tak obrovské množství vstupů, že ani při společné práci většího množství testerů, by nebylo možno v reálném čase tento problém zvládnout. Ovšem čím více hodnot otestujeme, tím vyšší je pravděpodobnost, že v daném softwarovém produktu není chyba, případně, že chybu najdeme. Práce testera se ale z časových a finančních důvodů musí omezit pouze na určitou část z těchto možností. Naštěstí zde platí, že jestliže otestujeme jednu akci programu s určitými hodnotami a zjistíme, že program pracuje, nebo nepracuje správně, lze předpokládat, že stejná nebo podobná akce se stejnými, nebo podobnými hodnotami bude mít stejný výsledek.

Vzniká, ale další problém. Jaké hodnoty z celé množiny vybrat tak, abychom měli dostatečnou jistotu, že jsme nepřehlédli nějakou podstatnou chybu. Tento problém řeší tzv. testovací případy.

3.2.1 Testovací případy

Jedná se o množinu hodnot, které si tester zvolí jako efektivní k testování. Jako základ se vezme množina všech hodnot, které lze zadat do konkrétního místa v aplikaci a ty se pak musí zredukovat do omezené množiny, kterou je možno otestovat v čase vyhrazeném na testování.

Je zde ale podmínka. Výsledná množina by měla být tak obsáhlá, aby bylo možno s dostatečně velkou jistotou říci, že v aplikaci se nevyskytuje chyba. Existuje několik metodik pro volbu efektivních testovacích případů. Každá z nich detekuje některé chyby častěji a jiné může naopak přehlédnout. Proto je nejvýhodnější tyto techniky, na základě vlastního úsudku a zkušeností, kombinovat.

3.2.2 Vytvoření tříd ekvivalentních případů

Jednou z metod pro výběr vhodných testovacích případů, je vytvoření tříd ekvivalentních případů. Metoda spočívá v tom, že vezmeme určitou množinu vstupních hodnot a tu pak rozdělíme do menších množin, u kterých předpokládáme stejné výsledky a stejnou funkcionalitu programu. Ve výsledku pak již pracujeme s jedinou hodnotou z každé nově vytvořené množiny.

Máme-li například formulář, který požaduje napsání celého čísla, pak lze předpokládat, že stejného výsledku dosáhneme, pokud napíšeme číslo 3, 432, nebo 8479. Stejně tak u leteckého bitevního simulátoru můžeme vzít jako ekvivalentní případ, pokud lze v určité situaci zatočit s letadlem doprava i doleva. Pak lze předpokládat, že to bude možné i v jakékoliv jiné situaci. Tudíž pak celou množinu ekvivalence nahradíme otestováním jedné hodnoty z této množiny.

Jak je na první pohled vidět, tato metoda je velmi účinná a rozsáhle redukuje celkový počet možností. Posouzení, zda hodnoty mohou být brány jako ekvivalentní, je v některých případech

jednoduché, v jiných naopak nikoliv. Jak ovšem spolehlivě zjistit, co do množiny ekvivalentních případů patří a co ne? Rozhodnutí se dá poměrně lehce a přesně učinit na základě hraničních podmínek.

3.2.3 Hraniční podmínky

U velké většiny akcí se předpokládá zadání hodnot pouze v určitém rozsahu a určitého typu. Dalo by se říci, hodnot smysluplných. Například při zadávání jména do formuláře v internetovém obchodu se předpokládá, že uživatel zadá množinu znaků abecedy, při zadávání částky v účetním programu se předpokládá zadání celého čísla, u počítačové hry se předpokládá, že hráč se nebude pokoušet procházet zdí a podobně. Co se ale stane, pokud uživatel hranici, ať už záměrně, nebo náhodou, poruší. Každý kvalitní program by měl být na tyto situace připraven. Ať už zadá uživatel jakékoliv hodnoty, program by neměl přestat korektně fungovat a v lepším případě by měl uživatele informovat jaká hodnota, případně akce, se od něj očekávala.

Tester se pokouší všechny takovéto hranice v softwaru nalézt. Každá nová hraniční podmínka, znamená rozdělení třídy ekvivalence na dvě. Při testování hraničních případů je pak nejlepší otestovat všechny hodnoty přímo na hranici a dále pak alespoň jednu z každé třídy ekvivalence.

Nejběžnější hraniční podmínky, nad kterými by se měl tester zamyslet: „První/poslední, Začátek/konec, Prázdný/plný, Nejpomalejší/nejrychlejší, Největší/nejmenší, Nejbližší/nejvzdálenější, Minimum/maximum, Nad/pod, Nejkratší/nejdelší, Nejdřívější/nejnovější, Nejvyšší/nejnižší.“ [3]

3.2.4 Testování stavů - logika toku řízení softwaru

Většina aplikací má stavy, ve kterých se může nacházet. Stejně jako hodnot, které zadáváme do aplikace, tak i stavů, ve kterých se aplikace může nacházet je mnoho. Proto i v tomto případě uplatníme třídy ekvivalence.

Je také užitečné vytvořit si mapu stavů, což je grafické znázornění jednotlivých stavů programu a jejich přechodů. Někdy může být součástí projektové dokumentace, ale pokud není, může být výhodné si takovouto mapu vytvořit. Mapa stavů má obsahovat každý jedinečný stav, ve kterém se může software nacházet, podmínku, kdy přechází jeden stav v druhý a generovaný vstup nebo výstup při přechodu z jednoho stavu na druhý.

3.2.5 Testy splněním, testy selháním

Tyto dva přístupy se liší poměrně zásadně. Testy splněním se provádí zejména v prvních fázích vývoje. Jedná se o ověření funkčnosti podle specifikace, kdy zjišťujeme, jestli program dělá skutečně to, co má. Mají nicméně své místo také ve fázích pozdějších, když se ověřuje, jestli nově přidané funkce neovlivnily funkčnost starou. Zadáváme hodnoty, jež se dají očekávat, a snažíme se ověřit, zdali je implementace korektní.

Naproti tomu testy selháním se používají, když již máme ověřenou základní funkčnost testovaného programu nebo komponenty. Zde může tester popustit uzdu své kreativitě. Schválně se pak snaží volit hodnoty, které by mohly program dostat do kolizní situace. Kvalitní software by měl být připraven také na situace, které nejsou obvyklé. Tato testování kontrolují především jakost a uživatelskou přívětivost.

3.2.6 Statické a dynamické testování

Metody testování lze rozdělit do dvou kategorií, na statické a dynamické podle toho, zda potřebují ke svému spuštění mít funkční verzi testované komponenty, či ne.

Metody statického testování vycházejí ze skutečnosti, že nepotřebují testovaný kód spouštět. Tester musí mít detailnější znalost použitého programovacího, skriptovacího, nebo modelovacího jazyka a jeho struktur. Výhodou je, že začít využívat tyto postupy můžeme již v raných fázích vývoje, fakticky ihned, jakmile je sepsána specifikace. Mezi metody statického testování patří revize a validace kódu, nebo i analýza specifikace.

Dynamické testování k průběhu testu funkční verzi potřebuje. Tester porovnává chování za běhu s projektovou dokumentací.

3.2.7 Automatické a manuální testování

Testy můžeme také rozdělit podle toho, jestli je vykonává člověk, nebo počítač. Automatické testování má hned několik výhod. Nedochází při nich k selhání lidského faktoru a testy tak nejsou tímto směrem ovlivněny. V rychlosti automatické nástroje člověka také převyšují a to i několikanásobně. Jsou rovněž v mnoha případech levnější. Některé části testování však potřebují přítomnost lidského úsudku a není proto možné je automatizovat. A najdou se i takové, kde je z důvodu rozsahu jednodušší a rychlejší provést testování ručně. Nehledě na to, že i když práci vykonává počítač, lidská práce je stále nenahraditelná a automatické testy pouze usnadňují práci testerů.

V praxi se tyto dva způsoby kombinují. Člověk-tester, se soustřeďuje na vytváření a revizi testů, přičemž automatické nástroje za něj rychle a přesně tyto testy provádějí.

3.3 Druhy testování

Stejně jako existuje více metod testování, existuje i více oblastí, které je možno testovat.

3.3.1 Testování specifikací

Při testování specifikací se provádí revize zadání projektu. Hlavní účely tohoto počínání jsou především:

- ověření obsahové správnosti,
- snaha najít a upozornit na všechny části, které by se daly nejednoznačně pochopit,
- snaha vyjasnit a sjednotit terminologii,
- nalezení všech částí, na které se možná při tvorbě specifikace zapomnělo,
- nalezení všech částí, které neobsahují dostatek informací k jejich vytvoření.

Kvalitně vytvořená specifikace je k prospěchu obou stran, vývojového týmu i zákazníka. Vývojovému týmu by se měl produkt snáze tvořit. Na straně zákazníka je pak zajištěna spokojenost a může být garantována kvalita výsledného produktu. Obě strany pak mohou na tuto dokumentaci odkazovat v případě rozporů o funkčnosti, nebo podobě projektu.

3.3.2 Testy konfigurace

Testování konfigurace zjišťuje, zda je software schopen pracovat na hardwaru, pro který je určen. Protože konfigurace počítače, na kterém je aplikace vyvíjena, se málokdy shoduje s konfigurací počítače, nebo počítačů, pro které je produkt určen. Každá aplikace je závislá na hardwaru, který ji spouští, tudíž testy konfigurace jsou důležitým typem testování. Všechny projekty ovšem tento

druh testování bezpodmínečně nepotřebují. Záleží na posouzení konkrétního projektu.

3.3.3 Testy kompatibility

Stejně jako hardware, tak i operační systém, případně také software nainstalovaný v počítači ovlivňují práci a funkčnost softwaru. Proto i tento fakt musíme vzít v úvahu při testování. Vstupy a výstupy z programu by měly odpovídat standardům tak, aby se zajistila kompatibilita s jinými programy. Měla by být zajištěna funkčnost v různých verzích operačních systémů, případně i na různých platformách. Co bude otestováno, ovšem opět vychází z požadavků zákazníka, tedy z produktové dokumentace.

3.3.4 Testování cizích jazyků

U aplikací, které jsou lokalizovány ve více jazycích, nestačí pouze prosté přeložení. Musí se znovu zkontrolovat správnost, ale i funkčnost programu tak, jako by se jednalo o novou verzi softwaru. Nedostatkem přeloženého programu může být třeba jen grafický detail v podobě textu, který se nevešel do tlačítka, ale co když se stane, že tlačítko podle textu zvětší svou velikost a celý formulář se rozhází do zcela nepřehledné podoby. Jako příklad můžeme vzít i obyčejné webové stránky v českém jazyce, které pokud nejsou správně lokalizovány, nezobrazují korektně znaky s háčky a čárkami, text se tak stává špatně čitelný a nesrozumitelný. Mohou nastat ovšem i dosti závažnější problémy, jako je přetečení paměti, nebo chybná hodnota proměnné, závislá na jazyku. Tyto problémy by však měla odhalit již analýza implementace.

3.3.5 Testování použitelnosti

Při testování použitelnosti se především hodnotí celková kvalita softwarového produktu. A to z pohledu přehlednosti a přívětivosti uživatelského rozhraní. Ať už je totiž software jakkoliv sofistikovaný, vždy by se s ním mělo dát pracovat co možná nejjednodušeji. Uživatel by měl mít již při prvním spuštění pocit, že programu rozumí a že před ním nic „neskrývá“.

3.3.6 Testování dokumentace

Zahrnuje zkontrolování všech částí, které netvoří samotný produkt, ale přesto s ním nějakým způsobem souvisí. Můžeme zde zařadit veškerý marketingový materiál, texty, grafiku, obaly, manuál, výukové materiály, on-line nápovědu, ale třeba i šablony a vzorové soubory. Všechny tyto věci tvoří okolí produktu, které můžeme nazvat dokumentace. Tato dokumentace utváří pohled, jakým na produkt nahlížejí zákazníci a uživatelé. Je důležité, aby dobře korespondovala se směrem, kterým se produkt ubírá, vhodně jej doplňovala a především, aby neobsahovala chybné či nejasné informace. Dobře vypracovaná produktová dokumentace může pomoci produkt lépe prodat a také častokrát pomůže ušetřit náklady na pozdější podporu.

3.3.7 Testování webových stránek

V dnešní době má na internetu své zastoupení spousta firem, které nabízejí široký sortiment od služeb až po konkrétní výrobky. Forma internetových stránek od svého vzniku prošla velikými změnami a mnohé internetové aplikace se dnes svou složitostí již přiblížily aplikacím desktopovým. Přesto mají webové stránky svá specifika a mezi ně bohužel patří i častější výskyt chyb, než v jiných aplikacích. Je to způsobeno tím, že na internetu je obrovská konkurence. Doba na vývoj aplikace bývá minimální, náklady se snižují a testování se nevěnuje příliš velká pozornost. Z toho samozřejmě plyne snížení kvality. Na druhou stranu uživatelé mají vysoké nároky. Mohou

rychle porovnávat s konkurencí, která je v prostředí internetu skoro všudypřítomná. Pokud se na stránkách objeví chyby, může to mít závažný dopad na jejich návštěvnost.

Mnoho postupů je pro testování webových stránek podobných, jako pro ostatní aplikace. Většina koncepcí testování vychází z testování černé, případně šedé skříňky, kdy je testerovi znám zdrojový kód webové stránky a jeho součásti. Samotný obsah, tedy grafika a text, který vidí koncový uživatel, pak lze podrobit stejnému testování, jako dokumentaci. Důležitým faktorem pro webovou aplikaci také může být testování zátěže, čemuž se bude věnovat podrobněji následující kapitola.

3.3.8 Zátěžové testy

Zátěžové testy jsou svým způsobem jedinečné v tom, že je není možno provést bez automatického nástroje. Jejich hlavním účelem je nalezení hranic softwaru při splnění určitých podmínek. Tím myslíme například maximální počet připojených uživatelů, které dokáže systém obsluhovat, či minimální operační paměť, při které má program odpovídající odezvu. Můžeme rozlišit dva druhy zátěžových testů. Zátěžové testy prostředků, při kterých zatěžujeme aplikaci na stejném hardwaru tím, že zvyšujeme počet požadavků, které má testovaný program obsloužit za daný čas. A zátěžové testy, u kterých je zatížení aplikace v čase konstantní a omezujeme prostředky, jako jsou paměť, diskový prostor, rychlost procesoru (CPU), nebo rychlost síťového připojení.

Během zátěžového testování se hledají slabá místa aplikací, která při určitém zatížení neodpovídají požadavkům, které na ně klademe. Po odhalení těchto chyb se dají opravit různými způsoby. V kódu se hledají efektivnější, rychlejší, nebo paměťově méně náročné implementace. U databází se provádí doladění pomocí indexace, nastavení log souborů a podobně. U síťových aplikací může pomoci i nastavení serveru. A je tady samozřejmě i možnost zvýšení hardwarových požadavků.

Provádí se až po ukončení ostatních testování, když je k dispozici spustitelná podoba aplikace, či komponenty. Přesněji v době, kdy se již v softwaru nevyskytují žádné zásadní chyby, které by mohly zátěžový test ovlivnit.

3.4 Testovací nástroje

Testovací nástroje již v dnešní době patří neodmyslitelně k testování. Důvodů je mnoho, pokusíme se uvést několik nejdůležitějších. První podstatný důvod, proč použít testovací nástroj, je zajištění vysoké kvality testů. Člověk je omylný a zvláště v případně, kdy se opakuje stejné testování po několikáté, může se objevit i u zkušených testerů tzv. profesionální slepota a dojde k přehlédnutí chyby. Testovací nástroj z tohoto pohledu pracuje stále stejně. Dalším faktorem je zrychlení testování. Automatické nástroje pracují v porovnání s člověkem několikanásobně rychleji a lze je nechat spuštěné i v době, kdy tester nepracuje. V případě, že si testy uložíme, lze je opakovaně použít. Stává se, že některá již otestovaná část programu se testuje znovu, aby se ověřilo, že nové úpravy programu neovlivnily původní funkčnost. Právě v tomto případě testovací nástroje šetří mnoho času. Pouze dohledáme již vytvořený test a můžeme jej znovu aplikovat, eventuálně provedeme drobné úpravy. Síla automatizovaných nástrojů se také ukazuje u zátěžových testů. Zátěžové testy totiž navozují stavy, které by se běžně navozovaly jen velice těžko. Myšleno tím je, jak omezování systémových prostředků pro spuštěný program, tak i zatížení desítek, stovek, někdy až tisíců uživatelů.

Jak již bylo v textu uvedeno, práce testera je ovšem nenahraditelná. Je to on, kdo musí psát testovací scénáře a kontrolovat testy, ale testovací nástroje jeho práci zrychlují a zkvalitňují.

3.4.1 Nástroje pro management testování

Tyto nástroje se dají používat na všechny testovací aktivity po celou dobu životního cyklu softwaru. Jejich účelem je přehledně uchovávat a poskytovat informace o průběhu testování. Obsahují nástroje pro sledování defektů, které uchovávají záznamy o chybách, selháních a jiných nedostatcích. Dále mohou zahrnovat nástroje pro management požadavků, uchovávající popisy požadavků. Umožňují stanovení priorit a přesné rozdělení kompetencí mezi konkrétní pracovníky. Umožňují vytváření reportů o postupu testování, nástroje pro srovnatelnost výsledků testů, či nástroje pro kvantitativní analýzy. Což kupříkladu znamená, přehled spuštěných testů a přehled úspěšně ukončených testů.

3.4.2 Nástroje pro statické testování

Nástroje pro statické testování jsou většinou součástí vývojového prostředí, ve kterém je software vyvíjen. Mezi jejich funkce patří kontrola správnosti napsaného kódu, upozorňování na syntaktické a sémantické chyby. V mnohých případech umožňují předcházet chybám tím, že programátorovi našeptávají často používané segmenty, či umožňují přímo úseky kódu generovat. Mohou vést dozor nad dodržováním standardů. Sofistikovanější nástroje také částečně kontrolují funkcionalitu. Z výčtu je jasné, že s nimi tester většinou přímo nepracuje, ale tyto nástroje velice snižují chybovost při psaní kódu.

3.4.3 Nástroje pro specifikaci a návrh testů

Nástroje pro specifikaci a návrh testů umožňují na základě modelu softwaru vygenerovat testovací vstupy a k nim také očekávané výstupy. Šetří čas, který by byl potřeba k ručnímu napsání testovacích dat. Takto získaná testovací data přesně odpovídají specifikaci, ze které vycházejí. Vyhneme se tak nedostatkům, jež vznikají při jejich ručním psaní. Existují nástroje, které tato data přímo používají k testování, či je generují během probíhajícího testu.

3.4.4 Nástroje pro vykonávání a zaznamenávání testů

Nástroje pro vykonávání testů umožňují automatické, nebo poloautomatické provádění testovacích scénářů. Nástroj pracuje na základě uložených vstupů, očekávaných výstupů a pomocí skriptovacího jazyka, nebo uživatelského rozhraní umožňuje ovlivňovat testovací scénář. Součástí může být testovací postroj, který je schopen nasimulovat okolí testované komponenty tím, že během testu poskytuje vhodnou odezvu. Je výhodný v případech, kdy nemáme ostatní komponenty z jakéhokoliv důvodu k dispozici. Může to být výhodnější i v případě, kdy ostatní komponenty máme a potřebujeme přesněji lokalizovat a izolovat chyby, což nám předvídatelné a kontrolovatelné prostředí jednodušeji umožní. Nástroje pro zaznamenávání testů pak vytvářejí při každém spuštění testu protokol, graf či jiná data, která později slouží k rozboru a usouzení závěrů z testu.

3.4.5 Nástroje pro dynamické a zátěžové testování

Takovéto nástroje se zaměřují na testování softwaru za chodu. Některé chyby se projeví až dlouhodobým používáním, nebo také jejich zvýšeným zatížením. Při takovýchto testech se spíše než u jiných projeví nedostatečné doladění programů, přílišná alokace paměti, použití algoritmů s velkou časovou složitostí a podobné chyby. Nástroje pro dynamické testování prověřují, jak se chová program při běžném používání, zatímco nástroje pro zátěžové testy simulují činnost někdy až tisícovek, výjimečně i miliónů uživatelů, používajících danou aplikaci, databázi, síť, či server.

Dále pak existují monitorovací nástroje, které nepřetržitě analyzují a reportují používání určitého softwarového zdroje, který je v provozu. Zpětnou analýzou pak lze odhalit případné problémy, které vznikají.

3.4.6 Další nástroje

Druhů testovacích nástrojů je velké množství a ne všechny je možno zařadit do některé z předešlých kategorií. Ovšem testovací nástroje nemusí být to jediné, co lze využít při testování. Testeři mohou také používat programy, primárně určené k jinému účelu. K zaznamenávání průběhu testu může posloužit například tabulkový procesor, některý z textových editorů a případně i databáze.

Zdroje pro kapitolu 3: [3], [4], [5], [6], [7].

4 Zátěžové a výkonnostní testy

4.1 Typy zátěžových a výkonnostních testů

Druhů zátěžových testů je mnoho, záleží na tom, jaká kritéria si stanovíme. Uvedeme zde rozdělení těch nejdůležitějších druhů.

4.1.1 Zátěžové testy

Zátěžové testy jsou testy výkonnosti, které se zaměřují na ověřování či určování funkčních vlastností. Toto ověřování (určování) probíhá během zatížení softwaru. Zatížení bývá podobné tomu, které očekáváme během provozu aplikace.

Přínosem je především:

- ověření, zda testovaná aplikace je schopna dosáhnout požadavků, které na ni z pohledu zatížitelnosti klademe,
- detekce chyb funkčnosti pod zatížením,
- může nám napovědět, kde jsou v softwaru slabá místa, která by mohla dělat problémy.

4.1.2 Výkonnostní testy

Výkonnostní testy se stejně jako předchozí typ testů, zaměřují na ověřování či určování funkčních vlastností softwaru při zkoušce. V tomto případě se ovšem snažíme zatížit aplikaci postupně, až k jejím hranicím. Hranicí se myslí bod, ve kterém přestanou být splněny podmínky na aplikaci kladené.

Přínosem tohoto druhu testování je:

- že víme, kde se nachází hranice zatížitelnosti.

4.1.3 Testy hraniční zátěže

Tento typ testů se pokouší zatížit aplikaci až do chvíle, dokud aplikace nezhavaruje, či nepřestane reagovat. Má za úkol stanovit, za jakých podmínek u testované aplikace dojde k selhání. Když se tak stane, je důležité zaznamenat hodnotu, při které se tak stalo a také se pokusit najít indikátory, jež by mohly upozorňovat na hrozící pád, ještě dříve než přijde. Aplikace by se pak měla nastavit tak, aby těmto stavům předcházela, anebo pokud už chybový stav nastane, aby byl ošetřen nějakým, pokud možno přijatelným způsobem.

Tímto způsobem zjišťujeme:

- jestli nedochází při selhání systému ke ztrátě dat,
- nakolik se aplikace zpomalí, před tím, něž úplně zhavaruje,
- jaké indikátory mohou upozornit na hrozící selhání,
- s jakými stavy máme v případně zhavarování softwaru počítat.

4.1.4 Testy odolnosti

Tento druh zátěžového testování se snaží nasimulovat situaci, kdy se dlouhodobě lehce zvedne zatížení nad běžnou mez. Může se tak stát v případně, kdy se nenadále zvýší počet uživatelů, přistupujících k aplikaci, případně hardware, na kterém je aplikace spuštěna, bude potřeba využít také k dalším účelům.

Přínosem je:

- zjištění, jak se bude aplikace chovat při tomto typu zátěže,
- můžeme se připravit na stavy, které by mohly nastat.

4.1.5 „Bodcové“ testování

„Bodcové“ testování - spike testing je druh zátěžových testů, při kterých se snažíme, někdy i opakovaně, o krátkodobé zvýšení zátěže nad určitou mez.

Přínosem je:

- zjištění, jak se bude aplikace chovat při tomto druhu zatížení,
- můžeme se připravit na stavy, které by mohly nastat,
- zjistíme, jaké indikátory mohou upozornit na hrozící selhání.

4.1.6 Kapacitní testování

Souvisí s plánováním kapacity, kdy potřebujeme plánovat, jaké prostředky budeme potřebovat v případě, že se zvýší vytěžovanost aplikace na určitou mez.

Význam kapacitního testování:

- zjišťuje aktuální využití kapacity,
- umožňuje naplánovat potřebu hardwarových prostředků a to i vzhledem k financování.

4.2 Sledovaná kritéria

4.2.1 Doba odezvy

Doba odezvy je důležitým kritériem, které vypovídá o kvalitě aplikace. Je to doba mezi akcí, kterou provedl uživatel a očekávanou reakcí aplikace. V rámci jednoho testu rozlišujeme odezvu:

- nejrychlejší,
- nejpomalejší,
- průměrnou,
- pro percentil uživatelů, či požadavků.

4.2.2 Propustnost

Tato vlastnost určuje, jak velký objem dat musí systém zpracovat za určitou časovou jednotku, nejčastěji sekundu. Pro lepší pochopení uvedeme příklad:

- počet vykonaných operací za sekundu,
- objem dat za sekundu.

4.2.3 Využití zdrojů

Někdy je potřeba zjistit, nakolik aplikace zatěžuje hardware, na kterém je spuštěna, případně jaký hardware by byl pro aplikaci vhodný. Tyto parametry se zjišťují při zátěži, kterou předpokládáme při normálním provozu. Sledujeme různé parametry, jako například zatížení:

- procesoru,
- paměti,
- disku,
- sítě.

4.2.4 Maximální zátěž

Pokud sledujeme maximální zátěž, zjišťujeme nejvyšší možné zatížení aplikace, při kterém ještě splňuje určité podmínky. Provádí se při aktuálním nastavení serveru a aktuální hardwarové konfiguraci.

5 Nástroje pro zátěžové testování

V této kapitole zhodnotím své vlastní poznatky, které jsem nabyl při testování se sedmi vybranými nástroji. Čtyři z nich HP Load Runner, IBM Rational performance tester, SilkPerformer a Wapt jsou komerčními produkty. Zbylé tři Apache JMeter, Clif a The Grinder jsou zdarma.

Při práci s jednotlivými nástroji jsem testoval webovou aplikaci, spuštěnou na serveru Apache 2.2.0 pod Windows XP. Tato aplikace byla vytvořena z šablony webového fóra PHPBB ve verzi 3.0.10 s nainstalovanou českou lokalizací. Pro své účely jsem ji obsahově naplnil texty z webové stránky www.csfd.cz. Počítač, na kterém jsem pracoval s testovacími nástroji, byl stejný, jako počítač, na kterém byl spuštěn již zmíněný server.

5.1 HP LoadRunner

5.1.1 Seznámení

Jedná se o profesionální řešení v oblasti zátěžových testů. Je to jeden z nejrozšířenějších a nejpoužívanějších nástrojů. Obsahuje velký počet zabudovaných monitorů a protokolů, jako kupříkladu HTTP (Hypertext Transfer Protocol) a HTTPS (Hypertext Transfer Protocol Secure), Web services, SAP (System Analysis and Program Development), Siebel, Oracle, ODBC (Open Database Connectivity), People Soft, TCP/IP (Transmission Control Protocol/Internet Protocol) a mnohé další.

Webové stránky produktu jsou přehledné. Je na nich celkové představení produktu, ale po některých informacích je potřeba trochu více pátrat. Po registraci a přihlášení se nám však otevřou další možnosti. Dostaneme se ke knihovně, obsahující soubory s dokumentací, která obsahují velké množství informací od představení nástroje a instalaci, přes obecné pojednání o zátěžovém

testování, až po konkrétní práci s HP LoadRunner.

Také mimo oficiální webové stránky HP (Hewlett-Packard) se dá najít velké množství informací a to především těch, týkajících se samotného testování. Ale většina vycházela z oficiální dokumentace.

5.1.2 Instalace

Hardwarová konfigurace doporučená pro verzi 11.00, je následující:

- minimálně CPU 1GB,
- 2GB RAM (Random-Access Memory),
- 2GB volného místa na disku,
- rozlišení obrazovky minimálně 1024x768.

LoadRunner Controler musí být nainstalován na počítači s operačním systémem Windows. Konkrétně Windows XP SP3 (Service Pack 3), Vista SP2, 7, Server 2003, nebo Server 2008. Dále je potřeba internetový prohlížeč Internet Explorer 6.0 nebo vyšší. Ale je odzkoušeno, že vše funguje bez problémů i s Mozilla Firefox ve verzi 10.0.2.

Pro operační systémy UNIX je podporována pouze instalace komponenty Load Generator, která generuje zatížení virtuálními uživateli.

HP Loadrunner je komerční produkt, na webových stránkách výrobce je ke stažení 10 denní zkušební verze. Pro její získání je potřeba se registrovat. Stažení instalačních souborů probíhá přes HP Download Manager, přístupný z webového prohlížeče. Což může být důležité, protože instalační soubory měly velikost několik gigabajtů (GB). Samotná instalace byla dlouhá, ale uživatelsky nenáročná.

5.1.3 Prostředí a tvorba scénáře

Od prvního spuštění HP LoadRunner působil jako velice profesionální produkt. Jako i další komerční nástroje nabízel možnost vytvářet testovací scénář formou tutoriálu. V počátcích se ovšem může stávat, vzhledem k rozsáhlosti produktu, že vytváření testovacího scénáře bude trvat déle než u ostatních komerčních testovacích nástrojů.

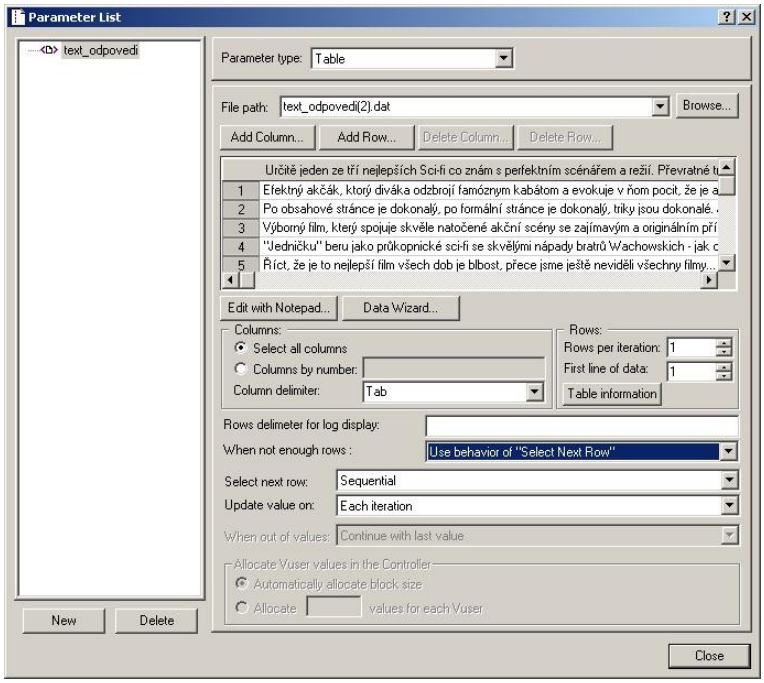
Přejděme ke tvorbě testovacího scénáře. Vytvoření nového skriptu se provádí vybráním možnosti „Create/Edit scripts“ v hlavní nabídce programu. Klikneme na vytvoření nového skriptu. Jako první nás čeká výběr testovaného protokolu a zvolíme „Web (HTTP/HTML)“. Po kliknutí na „Create“ se nám vytvořil nový prázdný testovací scénář. Pro tvorbu testovacího scénáře je možné využít pomocníka, který nás postupně provede všemi kroky tvorby testu. A proto i my dodržíme toto pořadí.

Jako první je na řadě nahrání skriptu v internetovém prohlížeči. Tester pracuje s testovanou webovou aplikací, jako normální uživatel, jehož činnost chce nasimulovat. Doladění testu se provede až v pozdějších krocích. Po ukončení nahrávání program provede verifikaci, čímž zkontroluje, jestli je možné nahraný scénář použít při testování. Protože některé akce, jako například opsání kódu z obrázku, program nasimulovat neumí.

Dalším krokem byly úpravy nahraného skriptu. Konkrétně přidání transakcí, parametrizace a kontrola obsahu stránek.

Přidání transakcí je dobrá volba v případě, že chceme některé akce seskupit do logických celků a tyto celky poté zohlednit při testu. Jako příklad lze použít přihlášení do aplikace, které se sestávalo z několika kroků. Kliknutí na „Přihlásit se“, zadání uživatelských údajů, odeslání údajů a nakonec kliknutí na obsah fóra. Tyto akce je možné seskupit do jedné transakce s názvem „Přihlášení“. Po tomto nastavení lze vlastnosti, jako chyby, časovou složitost a jiné, měřit pro celou transakci.

Parametrizace umožňuje testerovi přidat do testovacího scénáře prvek náhody, přiřazuje jednotlivým virtuálním uživatelům jedinečné hodnoty, a tím se snaží co nejvíce přiblížit testovací scénář běžnému provozu na serveru. V HP LoadRunner vytvoříme seznam parametrů, který následně použijeme ve skriptu jako proměnné. To znamená, pokud například chceme vytvořit test, ve kterém se bude přihlašovat deset nezávislých uživatelů, vytvoříme proměnnou „login“ s dvěma hodnotami „login“ a „password“, které naplníme hodnotami pro deset uživatelů. HP LoadRunner nabízí opravdu bohaté možnosti parametrizace (viz. obrázek číslo 3), a to jak ruční napsání parametrů, načtení ze souboru, ale třeba i načtení z databáze pomocí SQL (Structured Query Language). Stejně tak možnosti vybrání hodnot sekvenčně, náhodně, unikátně a mnohé další.



Obrázek 3: HP LoadRunner, možnosti parametrizace

Kontrola obsahu stránek je velice silný nástroj, který může naše virtuální uživatele „naučit samostatnosti“. Z virtuálního uživatele, který jen kliká podle předem určeného klíče, udělá uživatele, který je schopen na základě obsahu stránky rozhodnout, jak bude pokračovat. Možnosti zohlednění obsahu jsou opět velice široké. Jako příklad uvedeme: zohlednění na základě uživatelských hodnot, čísla iterace, náhodného čísla anebo času.

Na závěr vytváření skriptu je ještě možno provést některá nastavení, týkající se průběhu testu, ale tato nastavení lze provést i před samotným spuštěním scénáře.

5.1.4 Spuštění testu

Vrátíme se do hlavní nabídky programu a zvolením možnosti „Run Load Tests“ spustíme prostředí, ve kterém se spouští zátěžové testy. Po načtení tohoto prostředí nám program nabídne vytvoření nového scénáře. Vybereme námi vytvořený test. Nyní lze, pokud jsme tak ještě neučinili, nastavit logika testu, jako je počet iterací, či náhodný přístup. Dále nastavíme průběh testu s pohledu zatížení virtuálními uživateli.

Zajímavou možností je definice cílů testu, například odezva na transakce, počet chyb,

celková propustnost, průměrná propustnost a další. Program pak zadané hodnoty automaticky porovná s testovanými a v závěrečné analýze testera bude informovat o splnění, nebo nesplnění cílů testu. A následuje už jen spuštění testu.

Analýza se opět provádí v samostatném nástroji, který opět spustíme z hlavní nabídky. Otevřeme si provedený test a můžeme začít analyzovat. Jsme zde informování o splnění, či nesplnění cílů testu. Můžeme si prohlédnout grafy počtu uživatelů, propustnosti, odezvy, počtu přípojení a různých jiných. Další možností je nastavit velikost kroku grafu, nebo třeba porovnat dva grafy. Velice propracované je vytvoření reportů o testu, kdy si vybereme přímo, která data chceme z testu vybrat a vložíme je do něj.

Možnosti exportu dat z testování nejsou příliš široké, ale v tomto případě to není výrazný problém, protože analyzátor, který je součástí nástroje, obsahuje rozsáhlé informace o testu. Kdyby bylo potřeba data z testu exportovat, na výběr máme mezi exportem do formátu HTML (HyperText Markup Language) a do formátu tabulkového procesoru Microsoft Excel.

Zdroje pro podkapitolu 5.1: [10], [20], vlastní zpracování.

5.2 IBM Rational performance tester

5.2.1 Seznámení

Dobrá alternativa pro měření zátěže webových aplikací. Vývojáři z firmy IBM vsadili na známé prostředí Eclipse a nástroj implementovali do něj. Podporuje široké spektrum druhů aplikací, jako jsou HTTP, SAP, Siebel, SIP (Session Initiation Protocol,), TCP (Transmission Control Protocol) Socket a Citrix.

5.2.2 Instalace

Minimální nároky pro spuštění nástroje jsou:

- procesor 1.5 GHz Intel® Pentium® 4, nebo vyšší,
- paměť 1 GB RAM,
- 1.5 GB místa na disku a místo na další vývoj,
- rozlišení 1024 x 768, 256 barev, nebo větší.

Z pohledu operačního systému je, podporován nejen operační systém Windows, ale dokonce i Linux. Windows ve verzích Windows 2003, 2008, XP Professional, Vista a 7. Linux pak ve verzích Red Hat Desktop 4, Red Hat Enterprise Linux 4, Red Hat Enterprise Linux 5, SuSE Linux Enterprise Server Version 9.0, SuSE Linux Enterprise Desktop / Enterprise Server Version 10.0

Dále, pokud budeme chtít nahrávat scénář, potřebujeme internetový prohlížeč. Typ není přesně specifikován.

Je možnost si po zaregistrování stáhnout 30 denní zkušební verzi. Po jejím stažení, můžeme začít s instalací. Ta probíhá jako u všech komerčních produktů velmi jednoduše.

5.2.3 Prostředí a tvorba scénáře

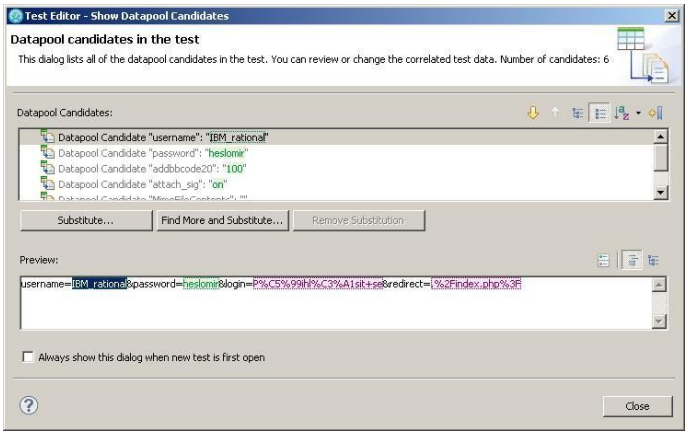
Tvorba scénáře probíhá ve vývojovém prostředí Eclipse, do něhož jsou integrovány funkce, umožňující zátěžové testování. Pokud začínáme, dá se k vytvoření testu využít předlohy, podle které lze postupovat tak, abychom nevynechali žádnou část nastavení scénáře.

Začneme vytvořením nového projektu „Performance Test Project“. Po jeho vytvoření se nám rovnou nabídne možnost nahrání prvního scénáře. Postupně navolíme základní nastavení, jako název a typ testu, použitý prohlížeč. Spustí se internetový prohlížeč, ve kterém postupujeme tak,

abychom při svém počínání „proklikali“ základní kostru budoucího testu. Po ukončení nahrávání je možno test ještě upravit ručním přidáním komponent. Na výběr máme komponenty, které zajišťují:

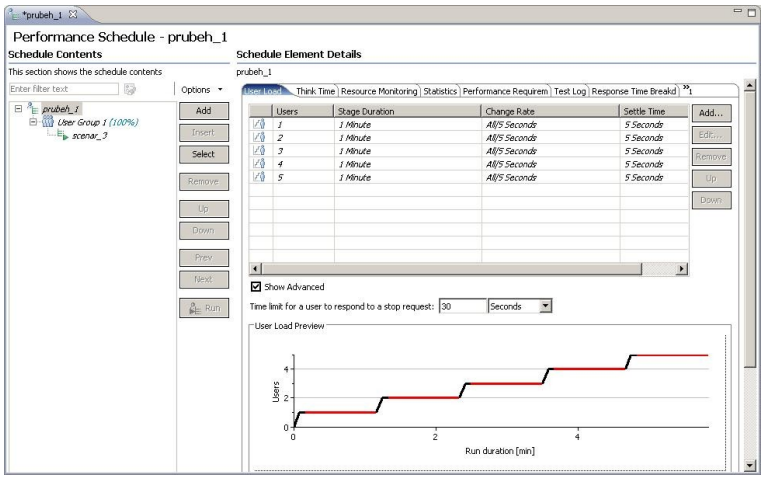
- volání - HTTP request,
- uživatelský skript,
- časové funkce – synchronizační bod, zpoždění,
- ovlivnění průběhu testu - smyčka, podmínka, náhodný výběr, transakce,
- komentář.

Po dokončení úprav scénáře jej necháme zkontrolovat a nástroj nabídne seznam kandidátů na proměnné, jak je uvedeno na obrázku 4. Což znamená, že nástroj v scénáři vyhledá položky, které by se mohly dynamicky měnit v závislosti na vlákně, jež bude scénář vykonávat.
Kromě výběru dat podle podmínky, patří mezi možnosti dosazování proměnných, i možnost načítání dat ze souboru.



Obrázek 4: IBM Rational performance tester: Možnosti parametrizace

Po upravení scénáře do požadované podoby následuje naplánování zatížení „Performance Schedule“. Možnosti nástroje v tomto ohledu jsou dobré a umožňují navodit všechny typy zátěže, která se běžně používá, viz. obrázek 5.



Obrázek 5: IBM Rational performance tester: Nastavení zátěže

5.2.4 Spuštění testu

Po spuštění testu se zobrazí grafy s daty naměřenými při testech. Jsou zde k dispozici, jak data týkající se softwaru, kupříkladu časy mezi odpověďmi a požadavky, naměřené během testu, tak i data, týkající se hardwaru jako: zatížení procesoru, čas odezvy procesoru, zatížení paměti, zatížení disku a podobně.

Nástroj obsahuje i funkce, které umožňují v rozboru dat zjistit, která místa aplikace jsou ta nejpomalejší. A to až do takového detailu, že lze dohledat problémová místa kódu. Takže po důkladném rozboru je schopen tester rozlišit původ chyby a navrhnout řešení, ve kterém může přímo popsat důvody a také kroky, které povedou k opravě problému.

Zdroje pro podkapitolu 5.2: [11], [21], vlastní zpracování.

5.3 SilkPerformer

5.3.1 Seznámení

Testovací nástroj od firmy Borland umožňuje vytvářet realistické zátěžové testy až tisíců uživatelů. Obsahuje poměrně velké množství monitorů a protokolů. Podporuje všechna důležitá prostředí Web 2.0 jako Adobe Flash/Flex, Microsft Silverlight a HTML/AJAX (Asynchronous JavaScript and XML).

Produktové stránky jsou přehledné. Z úvodní strany se uživatel i bez nutnosti přihlášení dostane k informacím, které by měly stačit odpovědět na hlavní otázky používání: „K čemu je určen?“, „Co umí testovat?“, „Jak nástroj vypadá?“, „Jak nástroj pracuje?“. Součástí je i přehledně zpracovaná FAQ (Frequently Asked Questions). Kladem je také kvalitně zpracovaný tutoriál, jenž obsahuje mimo jiné i odkaz na vzorovou webovou aplikaci, kterou je možno používat pro výuku testování.

5.3.2 Instalace

Minimální požadavky na hardware jsou:

- CPU Intel Pentium IV, nebo ekvivalentní, či vyšší,
- 512 MB RAM,
- 10 MBit Ethernet.

Z operačních systémů podporuje SilkPerformer pouze systém Windows a to ve verzích: 7, Vista, XP, Server 2008, Server 2003. Dále bude potřeba internetový prohlížeč. Jak je napsáno v dokumentaci, je potřeba Internet Explorer 5.0, nebo vyšší (doporučena je 5.5 nebo vyšší), či jiný prohlížeč.

Pro vyzkoušení produktu je k dispozici zkušební verze, s 30 denním klíčem. Její stažení je podmíněno vyplněním osobních údajů. Po stažení následuje poměrně dlouhá instalace, která ovšem probíhá podobně, jako běžné instalace.

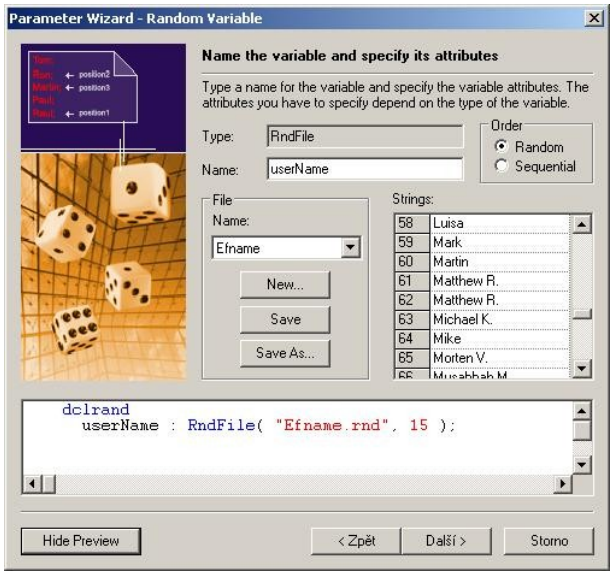
5.3.3 Prostředí a tvorba scénáře

SilkPerformer upoutal přívětivým testovacím prostředím, které se dobře používalo v prvních okamžicích, ale i později, když bylo potřeba prohlédnout různá rozšířená nastavení. Takže se s nástrojem bude dobře pracovat začátečníkům i pokročilým.

Průvodce vytvářením testu byl v tomto programu velice zdařilý. Tvorba testovacího scénáře začíná kliknutím na tlačítko „Start here“. Zadají se základní informace, jako název testu, popis

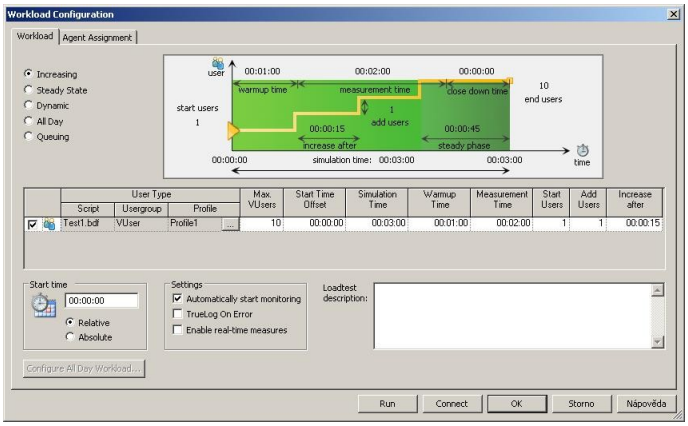
testu a vybrání typu testu. Následuje nahrání skriptu, kliknutím na „Model script“. Nahrání skriptu probíhá v internetovém prohlížeči. Program poté převede nahraný scénář do interního skriptovacího jazyka, ve kterém lze ještě provádět úpravy. Následuje zkouška skriptu. a pokud bude úspěšná, nabídne nám program možnosti úpravy a přizpůsobení.

Možnosti přizpůsobení se věnují připojení virtuálních uživatelů, nastavení prohlížeče a uživatelským parametrům. Tato část je méně propracovaná, než u jiných komerčních nástrojů. Ale naopak nastavení parametrizace bylo na srovnatelné úrovni a její integrace do zdrojového kódu byla přehlednější než u ostatních (obrázek 6). Parametry se daly vybírat náhodně, podle určitého klíče, nebo z předem připraveného seznamu hodnot, kde byly například uživatelská jména, adresy, poštovní směrovací čísla, jména měst a další. Samozřejmostí je výběr dat ze souboru.



Obrázek 6: SilkPerformer: možnosti nastavení uživatelských proměnných

Po přizpůsobení následovalo nastavení zátěže serveru. Zde se dalo vybrat z široké škály možností, od krátkých testů, po celodenní zátěž a od testů s konstantním počtem uživatelů, po přírůstkové a dynamické (obrázek 7).



Obrázek 7: SilkPerformer: možnosti nastavení zátěže

5.3.4 Spuštění testu

Když je připraven skript a jsou nastaveny možnosti zatížení, můžeme spustit provádění testu. Objeví se nám několik grafů, které ukazují, jak postupně probíhá test. V těchto grafech vidíme zobrazen v čase:

- počet aktivních uživatelů (vláken),
- počet chyb,
- počet aktuálních připojení,
- počet akcí za sekundu,
- aktivní odezvu,
- počet načtených stránek za sekundu.

Výstupem z provedeného testu může být například report, který si můžeme ve formátu „.html“ exportovat mimo program. V tomto reportu je vidět statistika celého testu. Grafy, které jsem již popisoval při průběhu testu jsou samozřejmostí. Součástí jsou dále tabulky rozložení odezvy, pro jednotlivé načítané stránky, z něhož se dá vyčíst, která část testovaného softwaru má při zatížení problémy.

Zdroje pro podkapitolu 5.3: [12], [22], vlastní zpracování.

5.4 Wapt

5.4.1 Seznámení

Wapt je nástroj pro testování výkonu a zátěže. Poskytuje snadno ovladatelný a cenově dostupný prostředek pro testování webu, od soukromých obchodních aplikací, přes webové servery, aplikační servery, až po databázová úložiště. Nabízí doinstalování modulů, rozšiřujících funkčnost, s nimiž podporuje i testování ASP.NET (Active Server Pages .NET), Silverlight, Adobe Flash a JSON (JavaScript Object Notation).

Webové stránky byly stavěné jinak, než u ostatních komerčních produktů. Dalo by se říci, že byly orientovány více na prodej nátroje, než na představení jeho konkrétních funkcí. Stránky byly více stručné, než u ostatních komerčních produktů, ale i tak obsahovaly množství důležitých informací. Součást stránek bylo fórum a také blog.

5.4.2 Instalace

Doporučená hardwarová konfigurace pro nástroj Wapt je:

- procesor Pentium 4/Athlon XP, nebo lepší,
- 2GB RAM,
- Gigabit Ethernet,
- 500MB volného místa na disku.

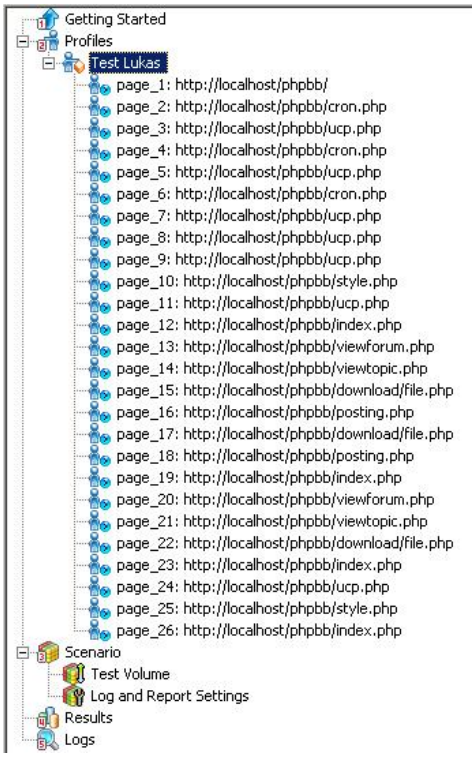
Softwarová konfigurace pak je operační systém: Microsoft Windows XP, 2003, Vista, 2008, nebo 7. A internetový prohlížeč Internet Explorer 6 či vyšší, Firefox 3.0 a vyšší, nebo Goole Chrome.

Stažení 30 denní testovací verze bylo naprosto jednoduché, bez nutnosti zadávání jakýchkoliv údajů. Velikost instalačního souboru byla oproti ostatním komerčním produktům nesrovnatelná, řádově několik MB, takže stažení bylo rychlé, stejně jako i následná instalace.

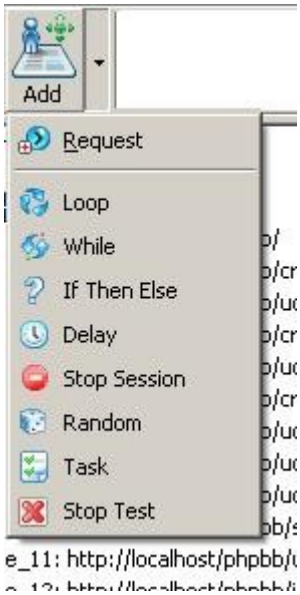
5.4.3 Prostředí a tvorba scénáře

Vývojáři tohoto produktu vsadili především na jednoduchost a rychlý vývoj testu. Zjednodušeně můžeme říci, že je sestaven ze tří kroků – nahrání scénáře, parametrizace a spuštění, přičemž všechny tyto kroky probíhají ve stejném prostředí.

Tvorba scénáře začíná vytvořením souboru „New scenario“. Dále se zvolí druh testu a poněkud nestandardně již v této fázi doba zátěže, a způsob zatížení. Následovalo nahrání skriptu v internetovém prohlížeči. Po tomto kroku se ještě daly ve scénáři (obrázek 8) provést jeho úpravy pomocí přidání, či odebrání komponent (obrázek 9).



Obrázek 8: Wapt: Detail scénáře

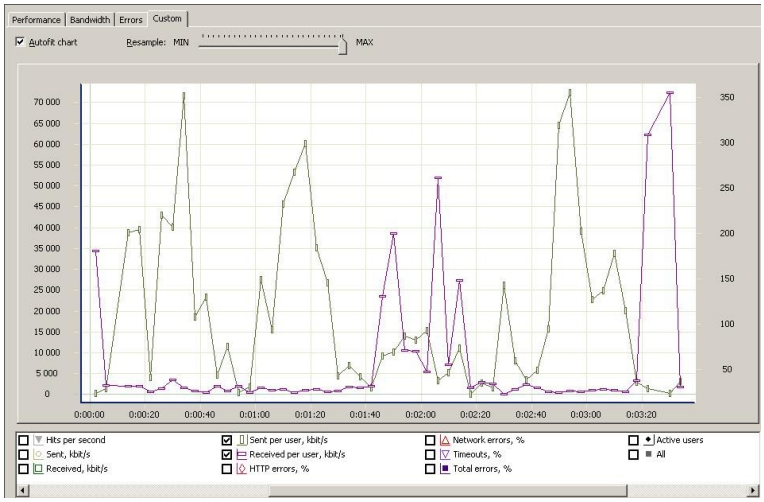


Obrázek 9: Wapt: Druhy komponent

Parametrizace probíhala tak, že se nastavily proměnné pro jednotlivé komponenty. Na výběr bylo opět z širokého výčtu možností, které byly až na vybírání hodnot z databáze podobné jako HP LoadRunner. Na závěr se pro test provedla verifikace, aby byla zajištěna jeho syntaktická bezchybnost.

5.4.4 Spuštění testu

Veškerá nastavení se provedla již při vytvoření testovacího scénáře a tak jen spustíme test.



Obrázek 10: Wapt: Graf zachycující průběh testu

Možnosti analýzy dat nejsou tak široké, jako u ostatních komerčních nástrojů. Hlavní indikátory, které se sledují u zátěžových testů však z grafů (obrázek 10) odhalit lze.

Zdroje pro podkapitolu 5.4: [13], [23], vlastní zpracování.

5.5 Apache JMeter

5.5.1 Seznámení

Apache JMeter je open source testovací nástroj, napsaný v jazyce Java. Slouží k testování aplikací pod zátěží a pro měření výkonnosti. Původně byl určen pouze pro testování webových projektů, ale od doby svého vzniku prošel značným vývojem a v dnešní době obsahuje i další funkce. Dá se tedy používat pro zátěžové testy Servletů, Perl skriptů, Java objektů, databází, FTP (File Transfer Protocol) serverů a dalších zdrojů.

Apache JMeter má ze všech neplacených testovacích programů na internetu nejlepší reference. A to například na stránkách apache.org, roseindia.net, wikipedia.org a mnoha dalších. Kromě toho najdeme na youtube.com i hodně videí, věnujících se práci v tomto nástroji, což se o ostatních open source nástrojích rozhodně říci nedá.

5.5.2 Instalace

Požadavky pro používání programu jsou JVM (Java Virtual Machine) minimálně ve verzi 1.5. Z pohledu operačního sytému omezení neexistuje. Požadavky pro Apache JMeter ve verzi 2.6 jsou přehledně v tabulce 1.

Operační systém	*JVMs*	*Architektura*
FreeBSD 9.0 (1)	OpenJDK 6 (2)	amd64
Linux 2.4	Sun JDK 5	1.1.386
Linux 2.6	Sun JDK 5, Sun JDK 6, Sun JDK 7, OpenJDK 6	i386, amd64
Linux 3.1	Open JDK 6, Open JDK 7	amd64
Mac OS 10.6.8	JDK 6	
Windows 7	Sun JDK 6, Sun JDK 7	32/64 bitů
Windows XP	Sun JDK 5, Sun JDK 6, Sun JDK 7	32 bitů

Tabulka 1: Požadovaná konfigurace pro verzi 2.6

1) BSD – Berkeley Software Distribution
2) JDK – Java Development Kit

Instalace programu se neprování, pouze nainstalujeme příslušný JVM, stáhneme archiv s programem a je možno začít s jeho používáním.

5.5.3 Prostředí a tvorba scénáře

Program disponuje přehledným uživatelským rozhraním. Při spuštění se již sám vytvoří nový testovací plán a tak je možno začít tvořit. Každý testovací plán obsahuje komponentu „Test plan“ (Testovací plán). V ní se dají nastavit uživatelské proměnné, které test používá při svém průběhu. Tyto proměnné fungují podobně, jako proměnné v běžných programovacích jazycích a mohou tak ovlivňovat průběh testu. Další komponentou, která v testu nesmí chybět, je „Thread Group“(Skupina vláken). Jedno vlákno si můžeme představit, jako jednoho uživatele individuálně vykonávajícího testovací plán. Tudíž počet vláken, rovná se počet simulovaných uživatelů. Lze nastavit počet spustitelných vláken, časový rozestup mezi spuštěním jednotlivých vláken a počet opakování jednoho vlákna. Je také možnost nastavit plánovač testů, který umožní zapnout test v určitý čas. V rámci vlákna pak již vkládáme komponenty, které řídí samotný průběh testu. Výčet těch hlavních:

- „Logic Controller“ - komponenty ovlivňující průběh scénáře, jako jsou cykly, rozhodovací podmínky a náhodný přístup,
- „Config Element“ - provádějí nastavení proměnných,
- „Timer“ - časovače a synchronizační body,
- „Samplers“ - komponenty zodpovědné za odesílání požadavků na testovanou aplikaci,
- „Pre Procesors“ - nastavení akcí, které se provedou před komponentou typu „Samplers“,
- „Post procesors“ - nastavení akcí, které se provedou po komponentě typu „Samplers“,
- „Assertions“ - používají se ke kontrole odpovědi na vyslaný požadavek,
- „Listener“ - výstupní data z testu.

5.5.4 Spuštění testu

Po spuštění testu se začnou provádět akce, tak jak jsou uspořádány v komponentách scénáře. Přičemž komponenty typu „listener“ zobrazují data testu. A to vždy podle toho, kde jsou umístěny ve scénáři. Apache JMeter nabízí celou řadu různých druhů grafických výstupů, ale hodně z nich nemá moc velkou vypovídající hodnotu.

Zdroje pro podkapitolu 5.5: [14], [17], [24], vlastní zpracování.

5.6 Clif

5.6.1 Seznámení

Clif je platforma pro zátěžové testy a generování zátěže. Obsahuje také funkce pro měření využití zdrojů. Podporuje protokoly HTTP, FTP, SIP a jiné. Je implementován ve čtyřech uživatelských rozhraních, mimo jiné v prostředí Eclipse.

Webové stránky nástroje působí přehledným dojmem a návštěvník se rychle dostane ke všem informacím, které by mohl potřebovat. Hned z úvodní strany se dozvíme, že byl nástroj oceněn v roce 2007 „Lutece d'Or 2007 award“, cenou pro nejlepší open source projekt, vyvíjený velkou firmou. Dále zde pak máme přehled produktu a jeho základní funkce. V dokumentaci programu najdeme, kromě specifikace produktu, odkaz na instalační příručku „Quick Start manual“ manuál, obsahující rychlý přehled práce se základními funkcemi, dále pak uživatelský manuál, vše ve formátu „.pdf“. Je zde i několik on-line nápověd a různé odkazy na soubory z konferencí. Tudíž mohu říci, že dokumentace a nápověda je podrobně propracovaná.

Projekt je stále aktivní. Dobrým indikátorem je i fakt, že o nástroji Clif se zmiňuje poměrně

velký počet jiných webových stránek, jako například opensourcetesting.org, forum.softwaretestinghelp.com, či testertools.com.

5.6.2 Instalace

Minimální požadavky na hardware nejsou v dokumentaci produktu uvedeny. Program však pracuje bez problémů i na notebooku Acer Extensa 5235, který odpovídá konfigurací, slabším počítačům na trhu.
Požadavky na operační systém jsou:

- každá distribuce Linux založená na jádru 2.6,
- Apple MacOS.X, nebo novější,
- Windows XP, či novější.

Ze softwaru pak potřebujeme:

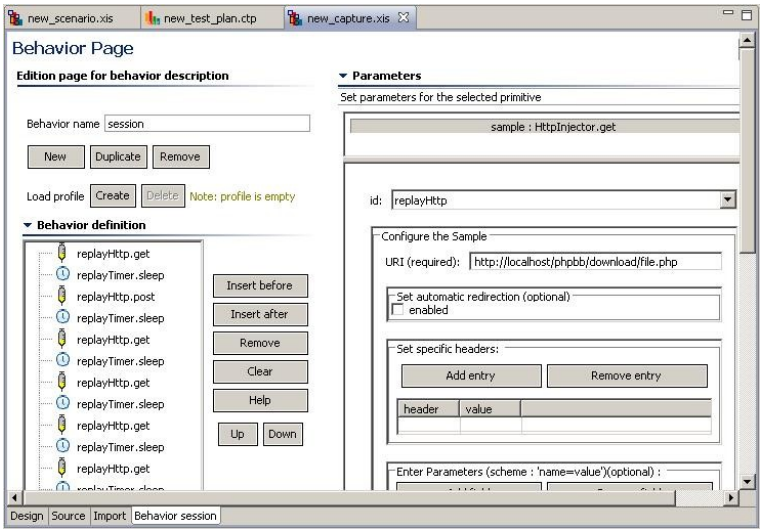
- minimálně Java 6 SE runtime, nebo Open JDK 6.0.,
- Apache veze 1.8.0, nebo vyšší.

Tento nástroj podporuje 4 různá uživatelská rozhraní, Swing GUI (Graphical User Interface), integraci do Eclipse, příkazový řádek a do Hudson/Jenkins integrovatelný plugin. Instalace se samozřejmě odvíjí podle toho, pro které rozhraní se rozhodneme. Způsoby instalace, či integrace jsou podrobně popsány v dokumentaci. K dispozici jsou ke stažení i verze Clif, u kterých je již vše připraveno. Dále bude uvedena práce s Clif integrovaným do Eclipse. V tomto případě stačí stáhnout zip archiv o velikosti asi 90 MB a může se začít s testováním.

5.6.3 Prostředí a tvorba scénáře

K vytvoření zátěžového testu se používá způsobu přidávání komponent do posloupnosti, podobně jako je tomu například v programu Apache JMeter. Dá se také použít nahrání testu ve webovém prohlížeči, plus jeho úprava.

Po načtení prostředí Eclipse, my vytvoříme nový projekt. V tuto chvíli se nabízí dvě možnosti, buď zvolit scénář „Isac scenario“, který se tvoří ručně, přidáváním komponent, anebo se můžeme rozhodnout pro „HTTP Capture“. Pokud zvolíme druhou možnost, můžeme vytvořit scénář automaticky, na základě akcí, které provedeme ve webovém prohlížeči. Pouze zvolíme možnosti nahrávání a Clif následně spustí proxy server. Stejně tak nastavíme i připojení našeho webového prohlížeče. Pak už jen stačí spustit nahrávání a ručně projít scénář, který chceme nechat nahrát. Výsledkem je seznam akcí, které jsme provedli v prohlížeči. V „Isac scenario“ bychom museli tyto akce vypsat ručně. A zde se dostáváme do bodu, kdy se oba přístupy opět spojily. Nyní máme ještě možnost projít si celý scénář a provést případné úpravy, jak je vidět na obrázku 11. Schopnosti programu z pohledu parametrizace jsou ale velice malé.



Obrázek 11: Clif: Tvorba scénáře

5.6.4 Spuštění testu

Nastavení spuštění je velice jednoduché. Navolíme počet uživatelů a jejich nárůst, nebo úbytek v čase. Poté již jen provedeme spuštění testovacího scénáře a můžeme sledovat několik druhů grafů, které nám zprostředkují výsledky testu. Jsou to konkrétně:

- průměrná, minimální, nebo maximální odezva,
- počet akcí v čase,
- propustnost akcí za sekundu,
- počet chyb,
- propustnost chyb za sekundu,
- poměr chyb v procentech.

Možnosti rozboru a analýzy testu jsou dosti nedokonalé, ale na zjištění základních hledisek, jako je počet chyb a výstrah, propustnost a odezva postačí.

Zdroje pro podkapitolu 5.6: [15], [26], vlastní zpracování.

5.7 The Grinder

5.7.1 Seznámení

The Grinder je open souce nástroj, vytvořený v programovacím jazyce Java. Je volně dostupný pod BSD-style open source licencí. Umožňuje za pomoci generického přístupu zátěžově otestovat jakýkoliv software, který má Java API (Application Programming Interface). Jako příklad lze uvést HTTP webové servery, webové služby SOAP (Simple Object Access Protocol), REST (Representational state transfer), aplikační servery a jiné. Integruje flexibilní skriptování za pomocí programovacího jazyka Jython, který je implementací programovacího jazyka Java v jazyce Python. Dále zahrnuje automatický management klientských připojení a cookies.

5.7.2 Instalace

The Grinder pracuje na kterékoliv hardwarové platformě a operačním systému, který podporuje J2SE 1.4 a vyšší. Pro své spuštění ve verzi 3 dále potřebuje Java Standard Edition 6.

Program není potřeba přímo instalovat, ale i tak musíme provést několik kroků. Jako první, ještě před spuštěním The Grinderu se musí nastavit CLASSPATH Javy, do které přidáme cestu k extrahované složce s programem, konkrétně ke grinder.jar. Spuštění konzole pak provedeme příkazem „java net.grinder.Console“.

5.7.3 Prostředí a tvorba scénáře

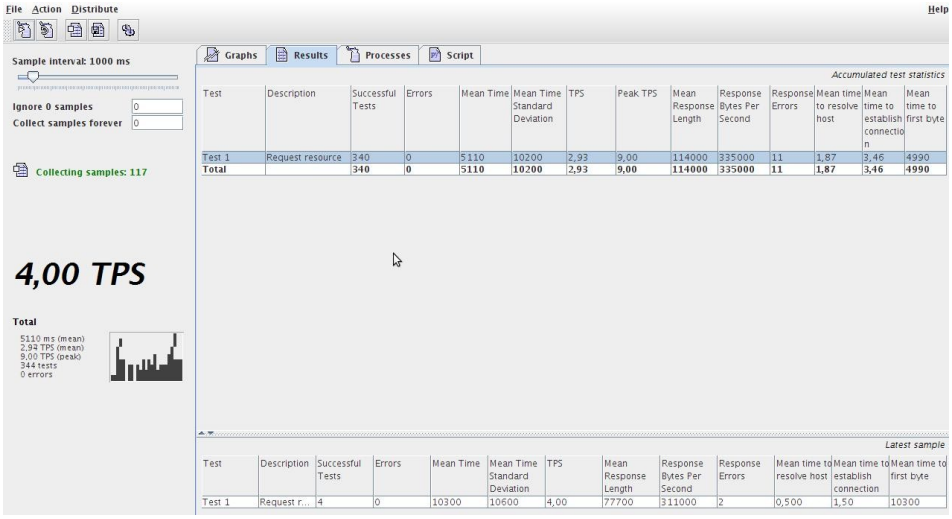
Průběh práce s nástrojem The Grinder je následující. V programovacím jazyce Jython napíšeme testovací scénář. Můžeme při tom využít knihovny nástroje, které obsahují objekty, jež zjednodušují práci s programem. V souboru grinder.properties nastavíme vlastnosti testu, vybereme soubor se scénářem k testování a spustíme test. Uživatelské rozhraní, které nám nabízí The Grinder je velice jednoduché a dá se použít vlastně jen jako spouštěč testů. Skript scénáře je jednodušší tvořit v nějakém textovém editoru, či vývojovém nástroji. The Grinder nabízí jednoduché prostředí pro správu a psaní Jython skriptů. Toto prostředí se ovšem nedá srovnat s jinými vývojovými prostředími. Nenabízí žádné funkce pro zjednodušení psaní kódu, ani zvýraznění syntaxe. Samotný kód testu je lépe napsat a odzkoušet v jiném vývojovém prostředí a již hotový jej přidat do projektu. Ovšem i přes tyto problémy je prostředí The Grinder výhodné použít, když chceme kód rychle prohlédnout, nebo editovat.

5.7.4 Spuštění testu

Způsob spuštění testu již byl naznačen v předchozí kapitole. Probíhá tedy tak, že nastavíme vlastnosti testu v souboru grinder.properties a spustíme test. V okně programu (obrázek 12) pak můžeme sledovat průběh testu. Je ale vidět jen několik základních informací:

- úspěšně/neúspěšně dokončené požadavky,
- provedené požadavky,
- průměrný čas,
- průměrný/maximální počet transakcí za sekundu,
- a některé další...

Takovýto výstup z testu bohužel moc nevypovídá a ve většině případů nebude dostačující. Při spuštění každého testování se zahájí vytváření dvou „.log“ souborů. Jeden, který se tvoří na základě požadavků, které necháme vypisovat z testu, a druhý obsahuje data z testu. Oba soubory jsou v textovém formátu, ale druhý soubor, obsahující data naměřená během testu, se dá celkem bez problému importovat do tabulkového procesoru, jako je Microsoft Excel, nebo OpenOffice Calc, což nabízí další možnosti analýzy.



Obrázek 12: The Grinder: Konzole v průběhu testu

Zdroje pro podkapitolu 5.7: [16], [27], vlastní zpracování.

6 Testování konkrétní webové aplikace

Tato část bakalářské práce popisuje čtrnáctidenní praxi ve firmě T-Mobile v Praze. Pro tuto praxi byl, firmou T-Mobile, přidělen konzultant pan Miloš Kalhous. Úkolem bylo zátěžově otestovat interní webovou aplikaci ve specifikovaných open source nástrojích. Výsledky se měly průběžně předávat konzultantovi. Ten na základně výstupních dat z testů prováděl nastavení a doladění serveru, na kterém byla spuštěna testovaná webová aplikace. Jelikož testy nebyly provedeny při stejných podmínkách, nelze provést komplexní porovnání jejich výsledků. Na závěr se mělo zjistit maximální zatížení, které testovaná aplikace dokáže zpracovat.

Testovaná aplikace sloužila k zadávání a zpracování požadavků v rámci firmy T-Mobile. Byla realizována formou fóra, ve kterém měl uživatel možnost prohlížet si a přidávat články, komentovat je a dávat jim hodnocení. Další informace vzhledem k tomu, že se jedná o aplikaci interní, nelze specifikovat. Některé části zobecním tak, aby se daly bez problémů publikovat. Přesto se domnívám, že tyto detaily nejsou, pro popis testování, podstatně důležité.

Prvním úkolem bylo vybrat dva open source testovací nástroje, ve kterých se mělo následně provést testování. Po konzultacích byly vybrány Apache JMeter a The Grinder. Důvodů bylo hned několik. Oba programy měly kvalitně zpracované webové prezentace, včetně dobře zpracované dokumentace. O vyšší jakosti nástrojů vypovídaly i dobré reference na internetu. V neposlední řadě konzultant tyto programy hodnotil velice dobře.

Testování se uskutečnilo v prostředí operačního systému Linux, distribuci Ubuntu. Nástroj Apache JMeter byl ve verzi 2.4 a The Grinder ve verzi 3.4.

6.1 Testovací scénář

Každé testování se řídí testovacím scénářem. Tudíž další krok po vybrání nástrojů pro testování, byl sestavení testovacího scénáře. Aplikace jako celek se, pro účely vytvoření scénáře, rozčlenila na posloupnost stavů, ve kterých se pak virtuální uživatel pohyboval a vytvářel tak aktivitu, která byla podobná reálnému uživateli. Vzhledem k dosažení větší přehlednosti se mohly některé stavy sloučit. A to takové stavy, které by samy o sobě neměly význam, nebo stavy, které po sobě vždy

logicky následovaly.

Po tomto zjednodušení tedy v testované aplikaci mohly nastat stavy, které nalezneme v tabulce 2, přičemž některých bylo možno dosáhnout pouze, pokud byl uživatel přihlášen. Tato informace je také obsažena v již zmíněné tabulce 2.

Stav	*Nutnost být přihlášen*
Přihlášení	-
Odhlášení	ano
Zobrazení profilu	ano
Změna hesla	ano
Prohlížení/čtení kategorie	ne
Prohlížení/čtení článku	ne
Prohlížení/čtení komentáře	ne
Přidání článku	ano
Přidání komentáře	ano
Hlasování	ano

Tabulka 2: Seznam stavů testované aplikace

Po stanovení a zjednodušení množiny jednotlivých stavů, se na jejich základě navrhlo několik „krátkých“ scénářů. Každý takový scénář se skládal z určité posloupnosti stavů a měl vždy jen jeden účel. To znamená, kupříkladu: jen přihlášení do systému, jen editaci článku a podobně. Jako příklad můžeme uvést přidání komentáře k článku. Uživatel jako první musí zobrazit úvodní stranu, poté provést scénář „Přihlášení do aplikace“, následuje zobrazení článku a seznamu komentářů, nakonec přidání komentáře.

Pro přehlednost se všechny „krátké“ scénáře zapsaly do tabulky. Jako vzor posloužila tabulka, kterou k těmto účelům v T-Mobile využívali. Tabulka se scénáři je uvedena v příloze A.

Na základně „krátkých“ scénářů se poté vytvořil jeden komplexní testovací scénář, jenž byl implementován v obou testovacích nástrojích. Mezi jednotlivými akcemi(requests) byly použity náhodně volené rozestupy tak, aby virtuální uživatel neprováděl akce moc rychle po sobě a přiblížil se tak reakcím reálného uživatele. Části scénáře byly následující:

- Anonymní prohlížení – uživatel si náhodně prohlíží jednotlivé články a komentáře. Počet prohlédnutých článků se pro konkrétní vlákno volí náhodně, stejně tak se i náhodně vybírá, který článek si virtuální uživatel prohlédne.
- Přihlášení – uživatel zadá a odešle přihlašovací údaje. Je 25% šance, že se uživatel splete při zadávání údajů a pak zkouší akci přihlášení znovu.
- Přidání komentáře – zadání předmětu a textu komentáře, poté odeslání údajů.
- Editace komentáře – zeditování náhodného počtu vlastních komentářů.
- Prohlížení – uživatel si prohlíží články a komentáře. Opět je použit náhodný přístup.
- Odhlášení – odhlášení z aplikace a ukončení vlákna.

6.2 XPath

Něž se dostaneme ke konkrétnímu testovaní, vysvětlíme funkci jazyka XPath (XML Path Language). Jazyk XPath slouží k adresování částí XML (Extensible Markup Language) dokumentů a protože i XHTML (Extensible HyperText Markup Language), HTML vychází z XML, je možné pomocí XPath jednoduše hledat v obsahu webových stránek určitý text. Podmínkou je ovšem validita stránek, která v tomto případě byla zajištěna.

Jedná se o způsob, jakým lze jednoduše virtuálního uživatele „naučit“, aby si v načtené testované stránce, vyhledal podle výrazu určitou její část. Tato část se poté může porovnat s předem definovanou hodnotou, jak bylo aplikováno v popisovaném scénáři. Pomocí výrazu se v kódu webové stránky našel konkrétní odkaz, či odkazy a podle definovaného pravidla se pak vybralo, na kterou stránku se uživatel přesune. Dá se tak poměrně jednoduchým způsobem simulovat činnost uživatele, který obsah načtené webové stránky vidí. Oproti reálnému uživateli má tu výhodu, že nehrozí možnost přehlédnutí určitého elementu stránky. Tento přístup má ale i své nedostatky. Jak již bylo uvedeno, problémy mohou nastat, pokud nejsou testované webové stránky validní. A problémem může být pochopitelně také chybně napsaný XPath výraz, který není v některých případech vůbec lehké přizpůsobit již vytvořeným webovým stránkám. Záleží tedy pouze na tom, jakým způsobem XPath výrazy použijeme. Ukázky aplikovaných výrazů při testování v programu Apache JMeter, jsou zobrazeny v tabulce 3.

Použití	*Výraz*
Pro zjištění session	//div[@id='header_login']//input[@name='YII_CSRF_TOKEN']/@value
Vyhledání tlačítka	//a[@id='ButtonText_4']/@href
Vyhledání odkazů	//a[@class=' textbutton forth']/@href
Vyhledání tlačítka	//div[@id='columnInfo']//span[@class='author']//a/@href
Vyhledání tlačítka	//a[@id='Menu_12_1']/@href

Tabulka 3: Příklady použitých XPath výrazů

6.3 Apache JMeter

Jako první se testování provedlo v nástroji Apache JMeter. Tento nástroj je totiž na rozdíl od nástroje The Grinder, vybaven přívětivým uživatelským rozhraním, a tak může pomoci začínajícímu testerovi proniknout do problematiky testování.

Program pracuje tak, že tester vytváří projekt jako chronologickou posloupnost komponent. Tyto komponenty obsahují, jak různé požadavky posílané na testovaný server, tak i komponenty ovlivňující běh testu a v neposlední řadě také tzv. posluchače, sloužící k zaznamenávání testovaných dat.

Při vytváření testovacího plánu se do nového projektu nejprve přidá povinná komponenta „Thread Group“ (skupina vláken), která zastupuje skupinu virtuálních uživatelů. Rozlišení uživatelů od sebe se provedlo pomocí komponenty „User Parameters“ (uživatelské parametry), ve které se nastavily přihlašovací údaje pro každého uživatele z množiny uživatelských jmen a hesel, které byly pro účel testování přiděleny. Dále se nastavila globální proměnná s náhodnými časovými rozestupy pro pozdější potřeby jejich hromadného nastavení. Poté se pomocí XPath výrazu získal

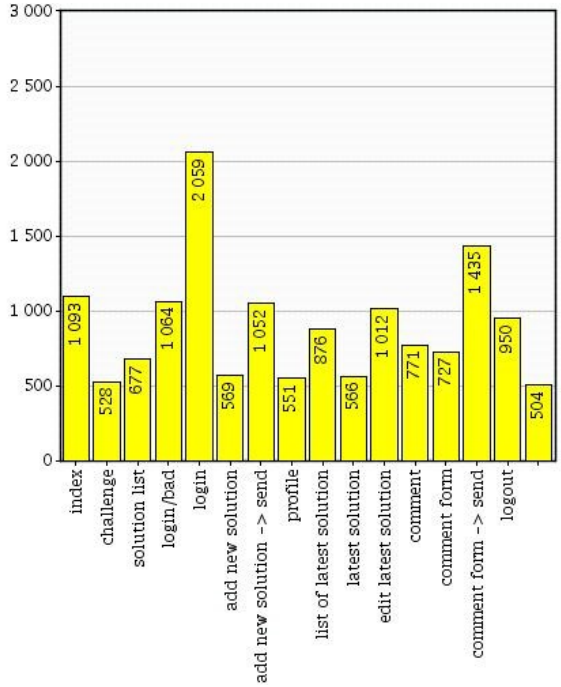
kontrolní řetězec, který posílal server v hlavičce HTML a nastavil se do cookies. V průběhu tvorby se některé komponenty vkládaly vícekrát, a proto jejich význam nebudeme znovu popisovat. Pro náhodný počet opakování některé části se použila komponenta „Loop Controler" (smyčka), v kombinaci s „Random Variable“ (náhodná proměnná). K výběru z určitých možností posloužil „Switch Controller“ (přepínač) v součinnosti s XPath výrazem z komponenty „XPath Extractor“. Na závěr celého scénáře se vložily komponenty typu „Listener“, jejichž účelem bylo zaznamenat průběh testu a přestože Apache JMeter měl poměrně hodně druhů těchto komponent, k účelům zaznamenání testu se použily pouze čtyři druhy. Podoba jejich výstupů je uvedena v příloze B.

Pro webovou aplikaci se provedly čtyři celé testy, plus několik zkušebních testů na počátku, aby se zjistilo, nakolik je možno testovanou webovou aplikaci zatížit. Hned prvním testem se totiž podařilo zahltit server, který poté přestal zpracovávat požadavky a bylo nutné jej resetovat.

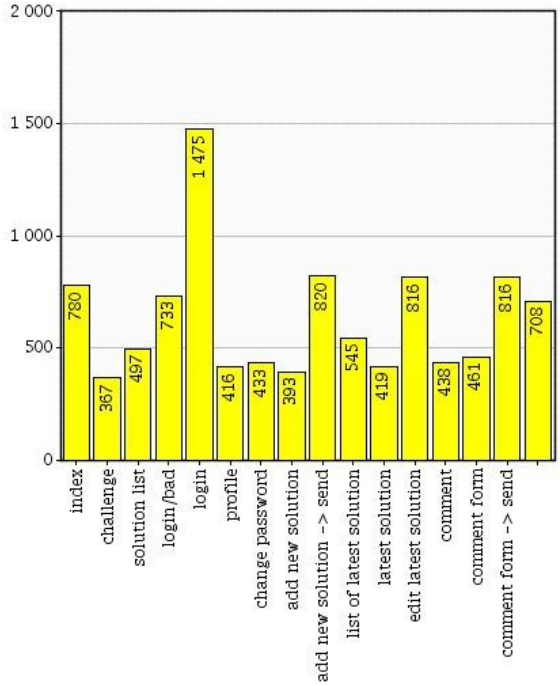
- Test číslo 1
  - 10 virtuálních uživatelů
  - průměrná odezva 816ms
  - celkem 1223 HTTP volání(requests)
- Test číslo 2
  - 15 virtuálních uživatelů
  - průměrná odezva 11913ms
  - celkem 1513 HTTP volání(requests)
  - některá volání skončila chybou
- Test číslo 3
  - 20 virtuálních uživatelů
  - průměrná odezva 1158ms
  - celkem 2413 HTTP volání(requests)
- Test číslo 4
  - 25 virtuálních uživatelů
  - průměrná odezva 579ms
  - celkem 3036 HTTP volání(requests)

Na prvním a druhém testu je vidět, že změna o pouhých 5 uživatelů zapříčinila zásadní změnu průměrné odezvy asi o 11 sekund. Před dalšími dvěma testy se už začal server ladit, takže průměrná rychlost odezvy zásadním způsobem klesla. Výsledky všech testů jsou v příloze C.

Na obrázcích 13 a 14 v závěru této podkapitoly se nacházejí pro srovnání dva grafy, první odpovídá testu s 10 uživateli a druhý testu s 25 uživateli. Je zcela patrné, že nastavení serveru Apache, na kterém byla webová aplikace spuštěna, testovanou aplikaci velmi zrychlilo a v závěru se povedlo dosáhnout lepších časů, při vyšším zatížení, než v počátcích při zatížení nižším.



Obrázek 13: Průměrná odezva [ms], při zatížení 10 uživateli před nastavením



Obrázek 14: Průměrná odezva [ms], při zatížení 25 uživateli před nastavením

6.4 The Grinder

Jak již bylo uvedeno, druhým použitým nástrojem byl The Grinder. V tomto nástroji se testovací scénář tvořil zcela odlišně. Před spuštěním bylo třeba učinit dva kroky. Nastavit soubor grinder.properties a naprogramovat skript testu. Výstupy zobrazuje program také jiným způsobem. Ve svém rozhraní sice zobrazuje průběh testu, ale tento výstup má spíše informativní charakter. Hlavní vypovídající hodnotu tak mají soubory „.log“ vytvářené u každého testu. Tyto „.log“ soubory mají textovou podobu a dají se tak jednoduchým způsobem importovat do různých programů, ve kterých provedeme analýzu.

Soubor grinder.properties je obyčejný textový soubor, který obsahuje veškerá nastavení testu. Možností je celá řada, většinu je možno nechat na implicitních hodnotách tím, že je nebudeme v souboru vyplňovat. Celý seznam hodnot k nastavení se dá najít v dokumentaci k programu. Části, které bychom neměli však opomenout vyplnit, jsou:

- grinder.script – cesta k souboru se skriptem scénáře,
- grinder.processes – počet procesů, které budou obsluhovat vlákna,
- grinder.threads – celkový počet vláken, tj. počet virtuálních uživatelů,
- grinder.runs – počet opakování testu,
- grinder.logDirectory – umístění, do kterého se mají uložit „.log“ soubory s výstupy,
- grinder.initalSleepTime – časové rozestupy mezi spuštěním jednotlivých vláken v milisekundách.

Příklad použitého nastavení grinder.properties:

```
grinder.script = testring.py
grinder.processes = 1
grinder.threads = 15
grinder.runs = 1
grinder.logDirectory = /home/milos/grinder-3.4/logs
grinder.plugin.parameter.followRedirects = true
grinder.plugin.parameter.useCookies = true
grinder.initialSleepTime = 1000
```

Vytváření testovacího scénáře se provádí výhradně v programovacím jazyce Jython. The Grinder obsahuje uživatelské rozhraní pro psaní skriptů, protože však tento editor nenabízí žádné možnosti usnadnění programování ani zvýraznění syntaxe, je dobré jej používat pouze k vybrání souborů a prohlédnutí jejich obsahu. Jako nástroj k editaci je vhodné využít služeb jiného programu. Bezplatných editorů, umožňujících zvýraznění syntaxe, nebo i našeptávání kódu je na internetu zdarma ke stažení celá řada a tak není problém využít možností některého z nich. V tomto případě byl použit editor gedit, který bývá běžnou součástí distribucí Linuxu.

V samotném skriptu se pak mohly používat v podstatě jakékoliv knihovny a samozřejmě knihovny nástroje. Ty obsahovaly především objekty jako:

- test – zaštiťující celý test,
- request – pro zasílání požadavků na server,
- response – pro ukládání odpovědí ze serveru,
- grinder – obsahující možnosti jako „logger.output“ pro zapisování informací do „.log“ souboru, či „sleep“ pro pozastavení činnosti vlákna.

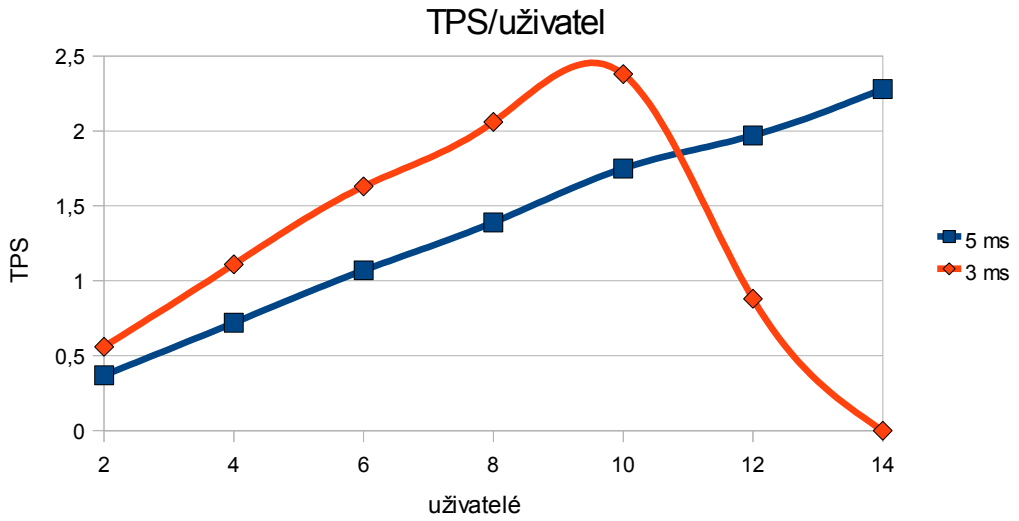
Celý skript testovacího scénáře je uveden v příloze D. Po jeho dokončení se mohlo přejít k jeho spuštění. Příklad použitého kódu, který obsahuje implementaci začátku testovacího scénáře:

```
#-----start
output("\n\n-----start as: \"" + user + "\", thead: " + str(thread_id) + "-----\n")

#-----index
output("-----index-----")
response = request.GET(url)
grinder.sleep(cas)

#-----browsing/anonymous in cycle
for a in range(0, (int(random_gen()) + 1) ):
  browsing()

#-----login/bad
if int(random_gen()) < 3:
  login_bad(user)

#-----login
login(user)
```

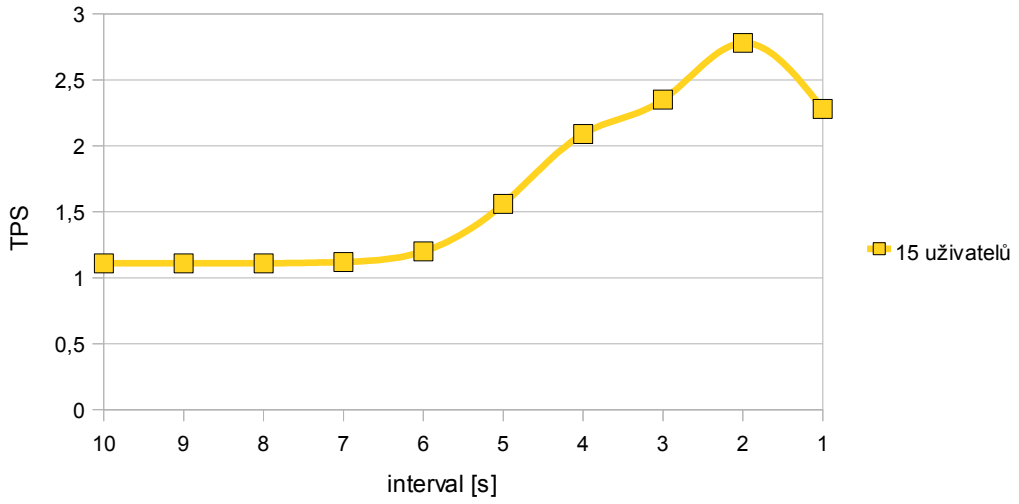
Po odladění serveru, které se provedlo na základě výstupů z nástroje Apache JMeter bylo úkolem najít maximální zatížení, na které dokáže server reagovat bez toho, že by došlo k jeho kolizi. Při tomto testování se používal stále stejný testovací scénář. Server se zatěžoval postupně se zvyšujícím počtem uživatelů, při stejných rozestupech mezi jednotlivými voláními v rámci vlákna tak, jak je vidět na obrázku 15. Při tomto testování s 12 virtuálními uživateli a rozestupem volání 3 sekundy, již jevil server jasné známky zahlcení a při následujícím testu se 14 virtuálními uživateli zhavaroval.

Další druh testování sledoval stejný parametr, jako test předchozí, ale zatížení se zvyšovalo postupným zkracováním intervalů, mezi voláními, při konstantním počtu uživatelů. Což je zobrazeno na obrázku 16. Z tohoto testu se ukázalo, že server zvládá maximální zatížení mezi 2,5 až 3 akcemi za sekundu.

TPS (Tests Per Second) jsou testy za sekundu. Což v tomto případě znamená průměrný počet požadavků za sekundu v rámci proběhlého testu.



Obrázek 15: Zatížení serveru při postupném zvyšování počtu uživatelů



Obrázek 16: Zatížení serveru při konstantním počtu uživatelů a postupném zkracování časových rozestupů mezi voláními

6.5 Srovnání nástroje Apach JMeter a The Ginder

Na závěr této kapitoly jsou oba nástroje porovnány podle toho, jak se s nimi pracovalo při tomto testování. Srovnávat se dalo velice dobře, protože každý nástroj přistupuje k testování diametrálně odlišně.

6.6 Instalace a spuštění

6.6.1 Apache JMeter

Před spuštěním je třeba mít nainstalován Java Virtual Machine minimálně ve verzi 1.5. Poté už stačí jen stáhnout archiv s nástrojem a spustit jej.

6.6.2 The Grinder

Zde prvnímu spuštění předcházejí o něco větší přípravy. Jako první se musí nainstalovat Java SE minimálně verze 6. Stáhneme archiv s nástrojem. Dále nastavíme proměnnou systému CLASSPATH na umístění, ve kterém se nachází adresář „lib“ nástroje. Program pak spouštíme z konzole příkazem „java net.grinder.Console“.

6.7 Vytváření scénáře

6.7.1 Apache JMeter

Tvorba scénáře probíhala v uživatelském rozhraní, které bylo poměrně snadno ovladatelné. Při tvorbě scénáře se dalo hodně věcí odvodit jednoduchou úvahou a nebylo potřeba často nahlížet do nápovědy. Pokud se ovšem bylo potřeba do nápovědy podívat, většinou se v krátkém čase hledaná informace našla. Menší problémy nastávaly, když bylo potřeba vytvořit složitější scénář. V té chvíli již tester musel předem promyslet, jak bude postupovat, aby později nemusel provádět rozsáhlejší změny. To totiž obnášelo projít jednotlivé komponenty a provést úpravy, což je jednak náročnější na čas a také náchylnější na chyby. Takže ve výsledku se dá říci, že pokud tester stanoví určitý systém ve své práci, je tvorba scénáře pomocí Apache JMeter jednoduchá a příjemná.

6.7.2 The Grinder

Scénář testu se v The Grinderu tvoří naprogramováním v jazyce Jython, který je velice podobný jazyku Python a se kterým se jednoduše pracuje. Vytváření testovacího scénáře se ovšem neobešlo bez hledání v dokumentaci. Na druhou stranu, když je tester obeznámen ze základními knihovnami, které při své práci potřebuje, otevře se mu široká škála možností úprav testovacího scénáře.

6.8 Možnosti analýzy výstupů

6.8.1 Apache JMeter

Apache JMeter má několik druhů výstupů. Jako nejdůležitější bych označil grafy s průměrným (minimálním, maximálním a poměrným) časem odezvy na požadavky, které se dají vložit do kterékoliv části testu. Můžeme tak dobře rozlišit, jaké vlastnosti jednotlivé části vykazují. Ale jsou tu i jiné možnosti vizualizace, jako výstupy do tabulky, kdy se zobrazují jednotlivé požadavky,

reakce na ně a další vlastnosti. Tyto možnosti by měly v mnoha případech stačit a pokud ne, je zde možnost nechat data o průběhu testu exportovat ve formátu tabulkového procesoru a tyto data pak dle libosti upravit.

6.8.2 The Grinder

The Grinder nabízí pouze jednoduchou vizualizaci testu, takže hlavní možností analýzy dat je vlastní úprava z „.log“ souborů, vznikajících při testech.

Zdroje pro kapitolu : [14], [16], [25], [28], vlastní zpracování.

7 Závěrečné shrnutí

Tato kapitola obsahuje porovnání nástrojů, které jsou zmíněny v této práci. Porovnávána jsou různá hlediska, jež mohou být důležitá při rozhodování o použití nástroje. Stupnice hodnocení je následující: nejlepší hodnocení je 10/10, nejhorší 0/10.

V další části textu se vyskytují zkratky jmen nástrojů a tak uvedeme jejich přehled (tabulka 4). Pořadí nástrojů je uvedeno tak, že nejdříve jsou podle abecedy seřazeny nástroje komerční a po nich také podle abecedy nástroje open source.

Celý název	*Zkratka*
HP LoadRunner	HP
IBM Rational performance tester	IBM
SilkPerformer	SP
Wapt	WT
Apache JMeter	AJM
Clif	CF
The Grinder	TG

Tabulka 4: Použité zkratky názvů nástrojů

7.1 Prezentace nástroje na internetu

V tabulce 5 je uvedeno porovnání, jakým způsobem se nástroje prezentují na internetu a jak působí na uživatele, kteří se rozhodují, zda nástroj použít.

Webové stránky, přehlednost – jde především o celkový vzhled internetové prezentace, orientace v ní při hledání konkrétních informací a rychlost dohledání informace.

Webové stránky, obsah – objem poskytnutých informací, při hledání konkrétního problému.

Dokumentace, přehlednost – přehlednost a orientace v dokumentaci, zhodnocení času, který je potřeba při hledání konkrétního řešení.

Dokumentace, obsah – rozsah informací v dokumentaci, vzhledem ke složitosti programu.

Kritérium	HP	IBM	SP	WT	AJM	CF	TG
Webové stránky, přehlednost	6/10	5/10	7/10	8/10	9/10	8/10	9/10
Webové stránky, obsah	6/10	6/10	8/10	7/10	9/10	9/10	9/10
Dokumentace, přehlednost	6/10	5/10	10/10	1/10	7/10	8/10	8/10
Dokumentace, obsah	10/10	9/10	9/10	1/10	9/10	10/10	9/10

Tabulka 5: Prezentace nástroje na internetu

Zdroje: [10], [11], [12], [13], [14], [15], [16], vlastní zpracování

7.2 Celkové možnosti nástroje

V tabulce 6 se nachází přehled celkových možností jednotlivých nástrojů z pohledu podpory technologií.

Technologie	HP	IBM	SP	WT	AJM	CF	TG
Webové Aplikace/HTTP	ano	ano	ano	ano	ano	ano	ano
Ajax/JavaScript	ano	ano	ano	ano	-	-	-
Adobe Flash/Flex	ano	-	ano	modul	-	-	-
Microsoft® Silverlight	ano	-	ano	modul	-	-	-
SOA (1)	ano	modul	-	-	-	-	-
Web services	ano	ano	-	-	-	-	-
RDP (2)	ano	-	-	-	-	-	-
Databáze	ano	-	ano	-	ano	ano	-
Citrix	ano	modul	ano	-	-	-	-
Java	ano	ano	ano	ano	ano	-	ano
.NET (NETwork)	ano	-	ano	modul	-	-	-
Aplikace Oracle	ano	modul	ano	-	-	-	-
SAP	ano	modul	ne	-	-	-	-
Monitorování HW (3)	ano	ano	ano	ano	-	ano	-
SIP	-	ano	ne	-	-	-	-
JSON	-	-	-	modul	-	-	-
SOAP	-	-	-	ano	ano	-	-
LDAP (4)	-	-	-	-	ano	ano	-
zSeries HW	-	modul	-	-	-	-	-

Tabulka 6: Možnosti nástrojů

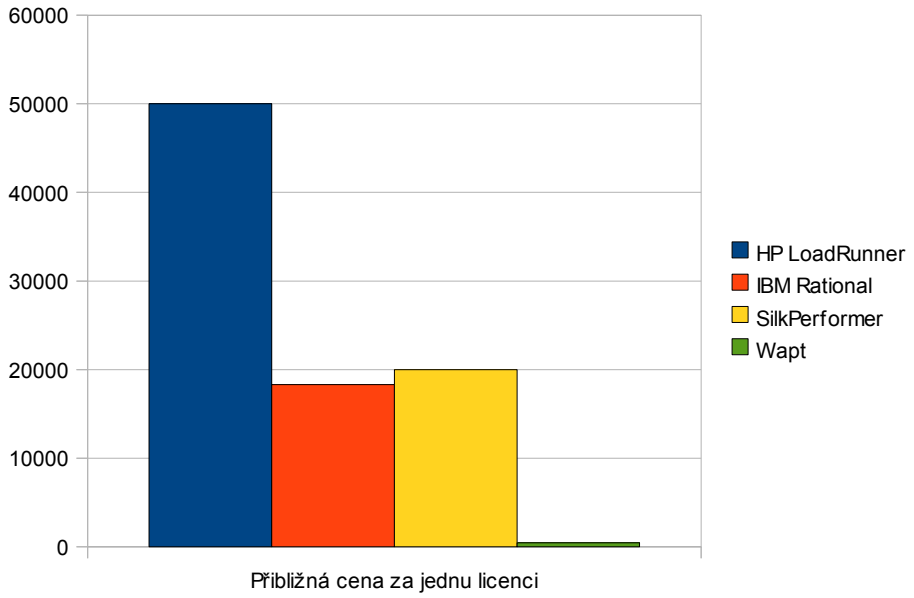
1) SOA – Service-oriented architecture

2) RDP – Remote Desktop Protocol
3) HW – Hardware
4) LDAP – Lightweight Directory Access Protocol

Zdroje: [10], [11], [12], [13], [14], [15], [16]

7.3 Cena nástroje

Důležitým faktorem je porovnání cen. Z tohoto pohledu jsou, na obrázku 17, porovnány pouze nástroje komerční. Bohužel některé firmy neposkytují příliš informací a tak se musíme spokojit s orientačními hodnotami. Ovšem i přesto je vidět, že cena jednotlivých nástrojů je diametrálně odlišná.



Obrázek 17: Orientační porovnání cen nástrojů [v USD]

Zdroje: [18], [19], [29], [30], vlastní zpracování

7.4 Platformní rozšířenost

V tabulce 7 jsou uvedeny požadavky nástrojů na operační systém.

Operační systém	HP	IBM	SP	WT	AJM	CF	TG
Windows	ano	ano	ano	ano	ano	ano	ano
Linux	pouze generátory zátěže	ano	-	-	ano	ano	ano
Mac OS X	-	-	-	-	ano	ano	ano
Solaris	-	-	-	-	ano	-	ano

Tabulka 7: Platformní rozšířenost

Zdroje: [10], [11], [12], [13], [14], [15], [16]

7.5 Výukové materiály

Porovnání nástrojů z pohledu přívětivosti pro začínající uživatele obsahuje tabulka 8.

Tutoriály – kvalita tutoriálů, videí a obecně výukových materiálů, které jsou k programu k dispozici, jejich vypovídající hodnota.

Pomoc během práce – na kolik nástroj vede začínající testery, míra jednoduchosti ovládání při prvním použití.

Nápověda programu – nápověda aplikace, která je dostupná během vývoje testu.

Kritérium	HP	IBM	SP	WT	AJM	CF	TG
Tutoriály	10/10	10/10	10/10	6/10	7/10	8/10	5/10
Pomoc během práce	8/10	8/10	10/10	8/10	4/10	2/10	0/10
Nápověda programu	10/10	7/10	9/10	8/10	8/10	6/10	5/10

Tabulka 8: Porovnání přívětivosti pro začínající uživatele

Zdroje: [10], [11], [12], [13], [14], [15], [16], [20], [21], [22], [23], [24], [26], [27], vlastní zpracování

7.6 Způsob tvorby testovacího scénáře

V této kapitole uvedeme způsoby tvorby testovacích scénářů, které jednotlivé nástroje implementují. V praxi se většinou používá kombinace různých způsobů, protože každý má svá specifika. Porovnání je v tabulce 9.

Nahrání pomocí webového prohlížeče – tento způsob vytvoření testovacího scénáře probíhá tak, že tester spustí z testovacího nástroje prohlížeč a následně provádí akce, které odpovídají práci reálného uživatele. Program zaznamenává práci, kterou tester provádí.

Výhody:

- relativně rychlý způsob,
- pro testera jednoduchý a nenáročný,
- není příliš potřeba znát testovanou aplikaci.

Nevýhody:

- scénář je nutno většinou ještě upravit,
- opakující se sekvence operací se musí provádět znovu.

Tvorba pomocí komponent – testovací scénář se tvoří jako posloupnost různých komponent, které vkládáme do chronologického pořadí tak, jak se mají provádět.

Výhody:

- tester má přehled o tom, co testovací scénář dělá.

Nevýhody:

- u větších scénářů je tato metoda zdlouhavá,
- je potřeba podrobněji znát testovanou aplikaci.

Napsání testu ve skriptovacím jazyce – scénář tvoříme programováním, nebo skriptováním v jazyce

Výhody:

- tester má přehled o tom, co testovací scénář dělá,
- velká variabilita,
- rychlá metoda.

Nevýhody:

- je potřeba podrobněji znát testovanou aplikaci,
- je potřeba znát programovací, či skriptovací jazyk nástroje.

Metoda	HP	IBM	SP	WT	AJM	CF	TG
Nahrání pomocí webového prohlížeče	ano	ano	ano	ano	ne	ano	ne
Tvorba pomocí komponent	ne	ano	ne	ano	ano	ano	ne
Napsání testu ve skriptovacím jazyce	ano (1)	ne	BDL (3)	ne	částečně (2)	XML	Jython

Tabulka 9: Porovnání nástrojů z pohledu tvorby scénáře

1) HP LoadRunner podporuje jazyk: C, Java, JavaScript a Visual Basic
2) Apache JMeter umožňuje vytvořit skript v rámci komponenty
3) Benchmark Description Language

Zdroje: [10], [11], [12], [13], [14], [15], [16], [20], [21], [22], [23], [24], [26], [27]

7.7 Možnosti parametrizace scénáře

Možnosti parametrizace scénáře obsahuje tabulka 10.

Data ze souboru – možnost načtení již připravených dat pro testování ze souboru.
Data z databáze – možnost načtení již připravených dat pro testování z databáze.
Nástrojem generovaná data – nástroj má připravená data, která může použít pro testování. Jako například seznam adres, jmen, názvů států a podobně.
Celkové možnosti – celkové možnosti parametrizace testovacího scénáře.

	HP	IBM	SP	WT	AJM	CF	TG
Data ze souboru	ano	ano	ano	ano	ne	ne	ano
Data z databáze	ano	ano	ano	ne	ne	ne	ano
Nástrojem generovaná data	ne	ne	ano	ne	ne	ne	ne
Celkové možnosti	10/10	9/10	8/10	4/10	4/10	3/10	9/10

Tabulka 10: Možnosti parametrizace scénáře

Zdroje: [10], [11], [12], [13], [14], [15], [16], [20], [21], [22], [23], [24], [26], [27], vlastní zpracování

7.8 Podpora distribuovaných testů

Pokud nástroj podporuje distribuované testy, je možno jej spustit několikrát na různých místech v síti, čímž se vytvoří cluster, který lze ovládat přes hlavní aplikaci. Tato schopnost nástroje je užitečná především v případě, chceme-li se při testu co nejvíce přiblížit reálné zátěži. Porovnání je uvedeno v tabulce číslo 11.

Nástroj	HP	IBM	SP	WT	AJM	CF	TG
Podpora distribuovaných testů	ano	ano	ano	ne	ano	ne	ano

Tabulka 11: Porovnání, podpora distribuovaných testů

Zdroje: [10], [11], [12], [13], [14], [15], [16]

7.9 Výstupní data

V tabulce číslo 12 nalezneme porovnání jednotlivých zátěžových testovacích nástrojů z pohledu možností, jakými lze pracovat s výstupními daty z testů.
Grafy – počet, druhy a vypovídající hodnota grafů.
Analýza – přehled možností analýzy testovaných dat.
Reporty – možnosti reportování testovaných dat z aplikace, sofistikovanost testů.

Kritérium	HP	IBM	SP	WT	AJM	CF	TG
Grafy	10/10	10/10	6/10	6/10	4/10	2/10	1/10
Analýza	9/10	10/10	7/10	4/10	4/10	4/10	1/10
Reporty	10/10	10/10	8/10	8/10	0/10	0/10	0/10

Tabulka 12: Porovnání možností výstupů z testů

Zdroje: [10], [11], [12], [13], [14], [15], [16], [20], [21], [22], [23], [24], [26], [27], vlastní zpracování

7.10 Celkový souhrn

7.10.1 HP LoadRunner

HP LoadRunner je absolutně nejrozsáhlejší zátěžový testovací nástroj na trhu. Nabízí široké možnosti, od fází vytváření testu až jeho spuštění a analýzu. Nachází uplatnění u velkých projektů, pro které je spolehlivým řešením.

Výhody:

- rozsáhlé možnosti při tvorbě testovacích scénářů,
- podpora několika programovacích jazyků,
- široké možnosti analýzy výstupních dat,
- přestože se jedná o nástroj rozsáhlý, nezatěžuje testera velkým množstvím nastavení.

Nevýhody:

- cena nástroje, která se pohybuje v miliónech Kč,
- některá nastavení mohou být pro začátečníka nejasná.

7.10.2 IBM Rational performance tester

V rozmezí několika posledních let se nástroj IBM Rational performance tester stal hodně používaný. Především proto, že má široké možnosti při tvorbě scénářů a nižší pořizovací cenu v porovnání s podobně propracovanými programy.

Výhody:

- porovnání kvality a ceny,
- rychlost dohledání možných problémů v testované aplikaci.

Nevýhody:

- prostředí Eclipse může být pro začínajícího testera nepřehledné.

7.10.3 SilkPerformer

SilkPerformer je jeden z velkých testovacích nástrojů. Umožňuje tvorbu jednoduchých a rychlých testů, ale nemá obtíže ani s propracovanějšími scénáři. Nabízí propracované uživatelské rozhraní, které se velice dobře ovládá.

Výhody:

- jednoduchá a rychlé ovládání,
- široké možnosti při tvorbě scénáře.

Nevýhody:

- v porovnání kvality a ceny jej rozhodně předstihuje nástroj IMP Rational performance

tester.

7.10.4 Wapt

Testovací nástroj Wapt vsadil především na jednoduchost a nízkou cenu. Nenabízí tak široké možnosti jako ostatní komerční nástroje. Hodí se proto pro méně rozsáhlé komerční projekty.

Výhody:

- cena,
- jednoduchost práce s nástrojem.

Nevýhody:

- celkově menší propracovanost, než ostatní komerční nástroje.

7.10.5 Apache JMeter

Apache JMeter je rozhodně nejpoužívanějším z oblasti open source nástrojů. Vývoj tohoto programu stále postupuje, čímž se zvyšuje i jeho kvalita. Nabízí přívětivé uživatelské rozhraní. Testera při práci dobře vede a nabízí i široké možnosti při tvorbě testů. Je dobrou volbou i pro testování větších projektů.

Výhody:

- velké množství informací k tomuto nástroji,
- přehledné GUI.

Nevýhody:

- nepřítomnost tlačítka „Zpět“,
- se složitostí scénáře se zvyšuje čas na jeho úpravu.

7.10.6 Clif

Clif je jednoduchý open source nástroj, umožňující celkovou analýzu webové aplikace i serveru, na kterém je aplikace spuštěna. Umožňuje rychle tvořit jednoduché scénáře. Pokud však potřebujeme vytvořit složitější testovací scénář, je tento nástroj těžkopádný. Je vhodný především pro malé projekty.

Výhody:

- do detailu propracovaná dokumentace,
- jednoduchá obsluha.

Nevýhody:

- malé možnosti nástroje v oblasti parametrizace,
- při tvorbě scénáře se musí dávat pozor na případné chyby, protože se velice špatně dohledávají,
- nepříliš sofistikované výstupy z testu.

7.10.7 The Grinder

The Grinder je velice silným open source nástrojem. A to především díky programovacímu jazyku Jython. Je složitější na obsluhu, ale pokud hledáme nástroj, který nabízí široké možnosti, je dobrou volbou.

Výhody:

- používá programovací jazyk Jython a díky tomu nabízí velkou volnost při psaní testovacího plánu,
- součástí jsou knihovny, zjednodušující vývoj skriptů.

Nevýhody:

- obsahuje pouze velice jednoduché uživatelské rozhraní,
- absence prostředí pro psaní skriptů,
- tester musí umět programovat,
- potřeba znát detailněji testovanou aplikaci,
- absence prostředí pro analýzu výstupních dat.

8 Závěr

V teoretické části této práce je postupně uvedeno, jaký je význam testování v rámci vývoje softwaru. Postupně je vysvětlena problematika a postupy používané při testování. Uvedeny jsou jednotlivé druhy testů a také druhy nástrojů používané při testování. Další kapitola se pak věnuje konkrétněji problematice zátěžového testování. Dále se již uvádí způsob práce s jednotlivými nástroji. Zátěžových testovacích nástrojů je k dispozici celá řada, v této bakalářské práci je uvedeno jen několik vybraných. Výběr byl proveden na základně informací, které byly poskytnuty ze strany konzultantů z firmy T-Mobile a také informací vyhledaných k jednotlivým nástrojům na internetu.

Popis práce ve všech vybraných nástrojích je uveden v kapitole „Nástroje pro zátěžové testování„ a jsou zhodnoceny v kapitole „Chyba: zdroj odkazu nenalezen“. Dá se říci, že komerční nástroje obecně, jsou mnohem více propracované, než nástroje open source. Komerční nástroje většinou nabízí komplexní přístup k zátěžovému testování, oproti tomu nástroje open source řeší menší okruh problémů. Přesto lze tvrdit, že open souce nástroje, které jsou v této práci uvedeny se mohou použít i při testování některých firemních aplikací. V této bakalářské práci je uveden jen omezený počet testovacích nástrojů, z tohoto hlediska by proto bylo zajímavé, do tohoto porovnání zahrnout i některé další testovací nástroje.

V rámci praktické části byla zátěžově otestována konkrétní webová aplikace. Na základě testování provedeného v nástroji Apache JMeter se realizovalo ladění testované aplikace. V druhé části při testování nástrojem The Grinder se zhodnotilo, nakolik je možno testovanou aplikaci celkově zatížit.

9 Použitá literatura

[1] Prof. Ing. Ivo Vondrák CSc.: Úvod do softwarového inženýrství, 2002, URL: <http://vondrak.cs.vsb.cz/download/Uvod_do_softwaroveho_inzenyrstvi.pdf> [datum čerpání zdroje: 24.2.2011]

[2] Spirálový model, URL: <http://testovanisoftwaru.cz/manualni-testovani/modely-zivotniho-cyklu-softwaru/spiralovy-model/> [datum čerpání zdroje: 19.4.2012]

[3] Ron Patton: Testování softwaru, Computer Press, 2002, ISBN 80-7226-636-5

[4] ISTQB – International Software Testing Qualifications Board: Foundation Level Syllabus, Verson 2011, URL: <http://www.istqb.org/downloads/viewcategory/16.html> [datum čerpání zdroje: 28.4.2011]

[5] Mgr. Anna Borovcová: Testování webových aplikací, URL: <http://www.poeta.cz/Zaklady_testovani.pdf> [datum čerpání zdroje: 5.5.2011]

[6] Lee Copeland: A Practitioner's Guide to Software Test Design, Artech House, 2004, ISBN: 978-1580537919

[7] Glenford J. Myers: The art of software testing, Second edition, Word Association, Inc., 2004, ISBN 0-471-46912-2

[8] Software Testing - Performance Testing Types, URL: <http://www.testinggeek.com/software-testing-performance-testing-types> [datum čerpání zdroje: 20.4.2012]

[9] Zátěžové testy software, URL: <http://testovanisoftwaru.cz/tag/zatezove-testy/> [datum čerpání zdroje: 20.4.2012]

[10] Oficiální webové stránky nástroje HP LoadRunner, URL: <http://www8.hp.com/us/en/software-solutions/software.html?compURI=1169460> [datum čerpání zdroje: 15.4.2012].

[11] Oficiální webové stránky nástroje IBM Rational performance tester, URL: <http://www-01.ibm.com/software/awdtools/tester/performance/> [datum čerpání zdroje: 15.4.2012].

[12] Oficiální webové stránky nástroje SilkPerformer, URL: <http://www.borland.com/us/products/silk/silkperformer/> [datum čerpání zdroje: 15.4.2012]

[13] Oficiální webové stránky nástroje Wapt, URL: <http://www.loadtestingtool.com/> [datum čerpání zdroje: 15.4.2012]

[14] Oficiální webové stránky nástroje Apache JMeter, URL: http://jmeter.apache.org/ [datum čerpání zdroje: 15.4.2012]

[15] Oficiální webové stránky nástroje Clif, URL: <http://clif.ow2.org/> [datum čerpání zdroje: 15.4.2012]

[16] Oficiální webové stránky nástroje The Grinder, URL: <http://grinder.sourceforge.net> [datum čerpání zdroje: 15.4.2012]

[17] Martin Hynar: JAVA – nástroje, Neocortex spol. s r.o., 2004, ISBN 80-86330-16-8

[18] SilkPerformer Selling Tactics vs LoadRunner, URL: <http://www.docstoc.com/docs/20503423/SilkPerformer-Selling-Tactics-vs-LoadRunner> [datum čerpání zdroje: 26.4.2012]

[19] Web Aplication Testing WAPT – Order, URL: <http://www.loadtestingtool.com/order.shtml> [datum čerpání zdroje: 26.4.2012]

[20] HP LoadRunner, [počítačový program]. Ver. 11.00, Hewlett-Packard, HP Software Division, 10 days evaluation version, automatizovaný nástroj pro testování zátěže

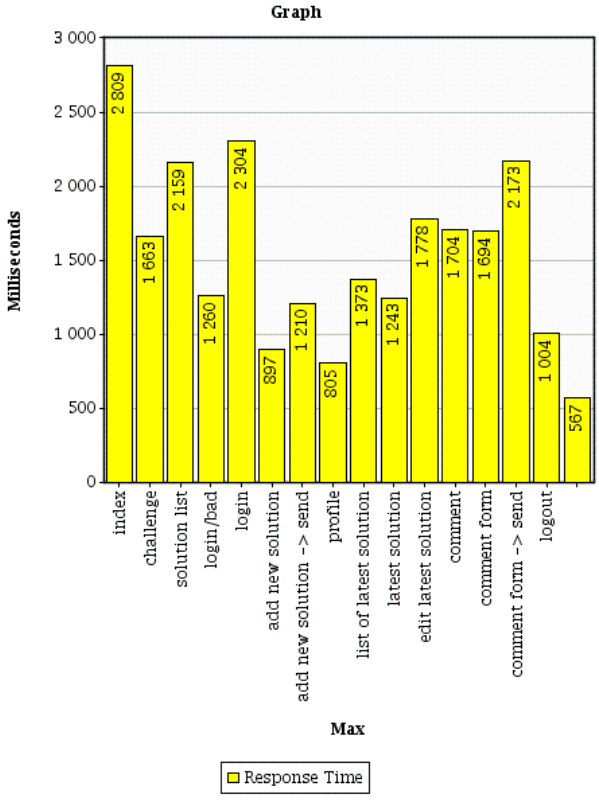
[21] IBM Rational Performance Tester, [počítačový program]. Ver. 8.2.1, IBM / Rational Software, Trial evaluation 30 days, Nástroj pro automatizované testování webových a serverových aplikací
[22] SilkPerformer, [počítačový program]. Ver. 8.3.0, Borland, Free 30-day trial, Nástroj pro zátěžové testování
[23] WAPT, [počítačový program]. Ver. 7.5, SoftLogica, 30 days trial, Zátěžový testovací nástroj
[24] Apache JMeter, [počítačový program]. Ver. 2.6, open source, Open source nástroj pro zátěžové testování
[25] Apache JMeter, [počítačový program]. Ver. 2.4, open source, Open source nástroj pro zátěžové testování
[26] Clif, [počítačový program]. Ver. 2.0.7, open source, Platforma pro zátěžové testy
[27] The Grinder, [počítačový program]. Ver. 3.8, open source, Open source nástroj pro zátěžové testování
[28] The Grinder, [počítačový program]. Ver. 3.4, open source, Open source nástroj pro zátěžové testování
[29] Cost of SilkPerformer licenses, URL: <http://www.sqaforums.com/showflat.php?Cat=0&Number=348059&an=0&page=2> [datum čerpání zdroje: 26.4.2012]
[30] Rational Performance Tester <http://www-142.ibm.com/software/dre/ecatalog/detail.wss?locale=en_US&synkey=Z798126X20866W20> [datum čerpání zdroje: 26.4.2012]

Příloha A: Tabulka scénářů

Název testu	Popis testu	Název kroku	Popis kroku	Očekávaná reakce	Komentář	Data 01	Data 02
01 – Přihlášení do aplikace	Přihlášení uživatele do aplikace. Zahrnuje také eventualitu, že se splete při zadávání svých udajů.	01 - Špatné zadání parametrů	Uživatel zadá špatné uživatelské jméno,a nebo heslo. Poté odešle požadavek na přihlášení.	Chybová zpráva: Chybné přihlašovací údaje. - uživatel nepřihlášen	http://testring.cz.tmo/	uživatelské jméno	heslo
		02 - Správné zadání parametrů	Uživatel zadá správné uživatelské jméno a heslo. Poté odešle požadavek na přihlášení.	Uživatel úspěšně přihlášen	http://testring.cz.tmo/	uživatelské jméno	heslo
02 - Přidání řešení		01 - Načtení hlavní stránky		Načetla se hlavní stránka	http://testring.cz.tmo/		
		02 - Přihlášení do aplikace	Scénář 01				
		03 - Zobrazení výzvy	Uživatel vybere konkrétní výzvu.	Zobrazení výzvy.	./index.php/challenge/detail/id_chl/29	id challenge z množiny: 15, 28, 29, 35, 36, 37	
		04 - Přečtení výzvy	Uživatel si přečte výzvu, což mu zabere 5-10 sekund a klikne na "zobraz řešení".	Zobrazení jednotlivých řešení.	./index.php/solution/solutionList/id_chl/29		
		05 - Zobrazení seznamu řešení	Uživatel klikne na tlačítko "nové řešení", pro přidání vlastního řešení.	Zobrazení formuláře, ve kterém uživatel napíše vlastní řešení.	./index.php/solution/solutionForm/id_chl/29		
		06 - Přidání vlastního řešení	Uživatel napíše své řešení v délce 50-250 znaků a odešle formulář.	Zobrazení jednotlivých řešení řešení. Nyní již i s vlastním řešením.	./index.php/solution/solutionList/id_chl/29	text řešení 50-250 znaků	
03 - Přidání komentáře		01 - Načtení hlavní stránky			http://testring.cz.tmo/		
		02 - Přihlášení do aplikace	Scénář 01				
		03 - Zobrazení výzvy	Uživatel vybere konkrétní výzvu.	Zobrazení výzvy.	./index.php/challenge/detail/id_chl/29		
		04 - Zobrazení seznamu řešení	Uživatel klikne na "zobraz řešení".	Zobrazení jednotlivých řešení.	./index.php/solution/solutionList/id_chl/29		
		05 - Zobrazení komentářů	Uživatel klikne na odkaz "přidej komentář" pod řešením.	Na stránce se zobrazí textové pole pro přidávání komentářů.	./index.php/comment/commentList/id_chl/29/id_sol/86/page		
		06 - Zapsaní komentáře	Do textového pole uživatel napíše komentář k řešení, dlouhý 10-100 znaků a klikne na tlačítko odeslat.	Aplikace zapíše komentář k řešení. A výsledek zobrazí v novém okně.	./index.php/comment/commentList/id_chl/29/id_sol/86/page	text komentáře 10-100 znaků	
04 - Hlasování		01 - Načtení hlavní stránky		Načetla se hlavní stránka	http://testring.cz.tmo/		
		02 - Zalogování do aplikace	Scénář 01				
		03 - Zobrazení výzvy	Uživatel vybere konkrétní výzvu.	Zobrazení výzvy.	./index.php/challenge/detail/id_chl/29		
		04 - Zobrazení seznamu řešení	Uživatel klikne na "zobraz řešení".	Zobrazení jednotlivých řešení.	./index.php/solution/solutionList/id_chl/29		
		05 - Hlasování	Uživatel hlasuje pro, nebo proti řešení.	Sloupce s hlasováním změní svou hodnotu.			

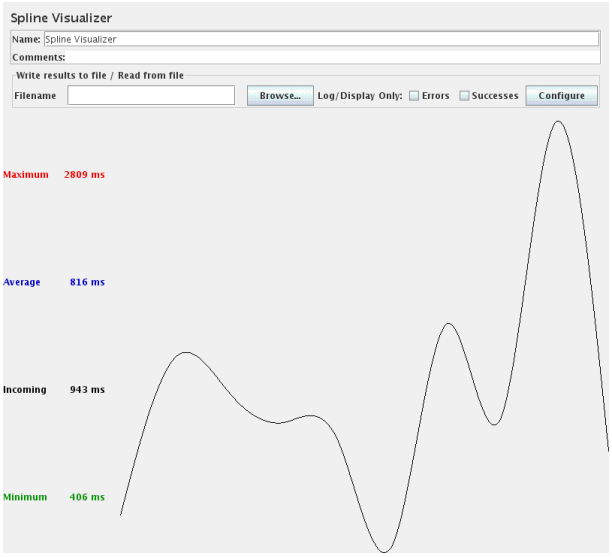
Název testu	Popis testu	Název kroku	Popis kroku	Očekávaná reakce	Komentář	Data 01	Data 02
05 - Změna hesla		01 - Přihlášení do aplikace	Scénář 01				
		02 - Ukázat profil	Uživatel klikne na odkaz se svým e-mailem pro zobrazení svého profilu	Zbrazí se profil uživatele.	./index.php/user/showProfile/id_usr/87		
		03 - Zobrazení formuláře změna hesla	Uživatel klikne na tlačítko "změnit heslo".	Zobrazí se nastavení hesla.	./index.php/user/updatePassword/id_usr/87		
		04 - Napsání nového hesla	Zapsání starého hesla a 2x nového hesla do polí a odeslání formuláře.	Zobrazí se zpráva. "Tvoje heslo bylo úspěšně změněno. ".	./index.php/user/updatePassword/id_usr/87	současné heslo	2x nové heslo
06 - Prohlížení řešení		01 - Načtení hlavní stránky		Načetla se hlavní stránka	http://testring.cz.tmo/		
		02 - Zobrazení výzvy	Uživatel vybere konkrétní výzvu.	Zobrazení výzvy.	./index.php/challenge/detail/id_chl/29	id challenge z množiny: 15, 28, 29, 35, 36, 37	
		03 - Přečtení výzvy	Uživatel si přečte výzvu, což mu zabere 5-10 sekund a klikne na "zobraz řešení".	Zobrazení jednotlivých řešení.	./index.php/solution/solutionList/id_chl/29		
		04 - Zobrazení seznamu řešení	Uživatel si přečte jednotlivá řešení, což mu zabere 5-10 sekund.				
07 - Prohlížení posledních komentářů		01 - Načtení hlavní stránky		Načetla se hlavní stránka	http://testring.cz.tmo/		
		02 - Zobrazení výzvy	Uživatel vybere konkrétní výzvu.	Zobrazení výzvy.	./index.php/challenge/detail/id_chl/29	id challenge z množiny: 15, 28, 29, 35, 36, 37	
		03 - Přečtení výzvy	Uživatel vybere konkrétní výzvu.	Zobrazení výzvy.	./index.php/challenge/detail/id_chl/29		
		04 - Zobrazení seznamu řešení	Uživatel klikne na "zobraz řešení".	Zobrazení jednotlivých řešení.	./index.php/solution/solutionList/id_chl/29		
		05 - Zobrazení komentářů	Uživatel klikne na odkaz "přidej komentář" pod řešením.	Na stránce se zobrazí textové pole pro přidávání komentářů.	./index.php/comment/commentList/id_chl/29/id_sol/86/page		

Příloha B: Použité výstupy programu Apache JMeter

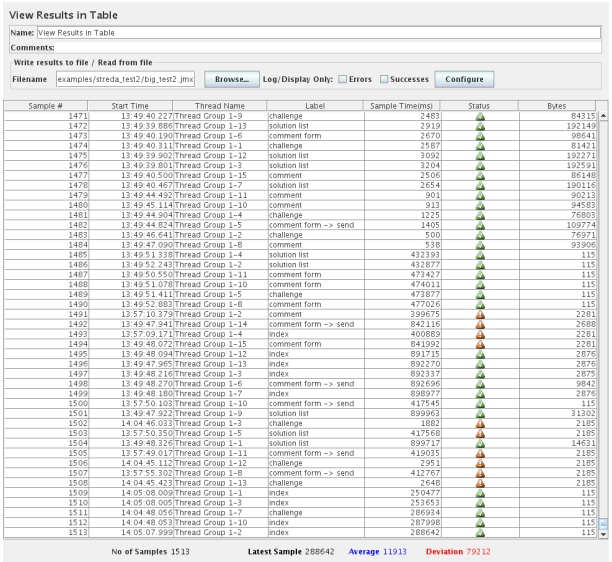


Výstup s programu Apache JMeter: sloupcový graf s odezvami

Výstup z programu Apache JMeter: časový průběh

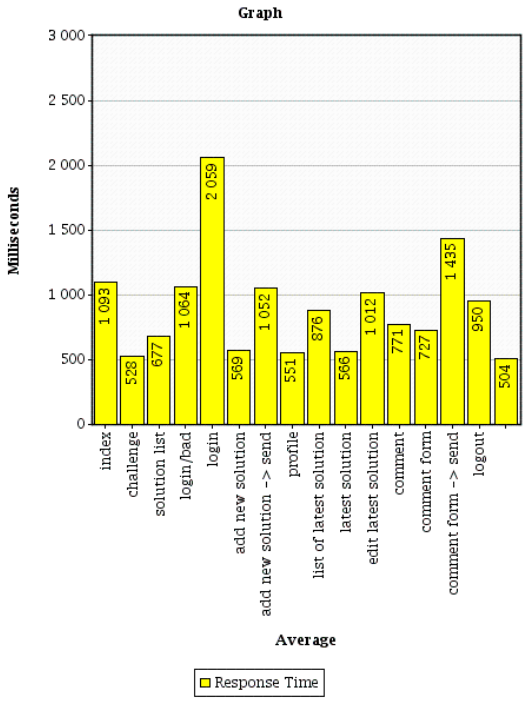


Výstup z programu Apache JMeter: rozdíly mezi odezvami

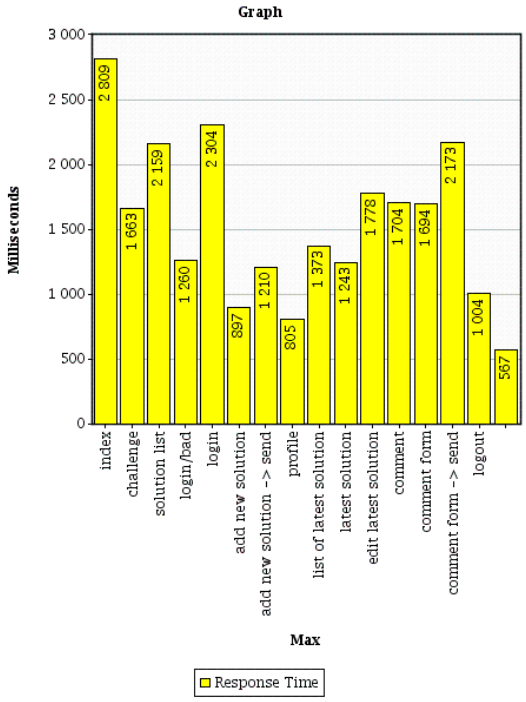


Výstup z programu Apache JMeter: odezvy programu

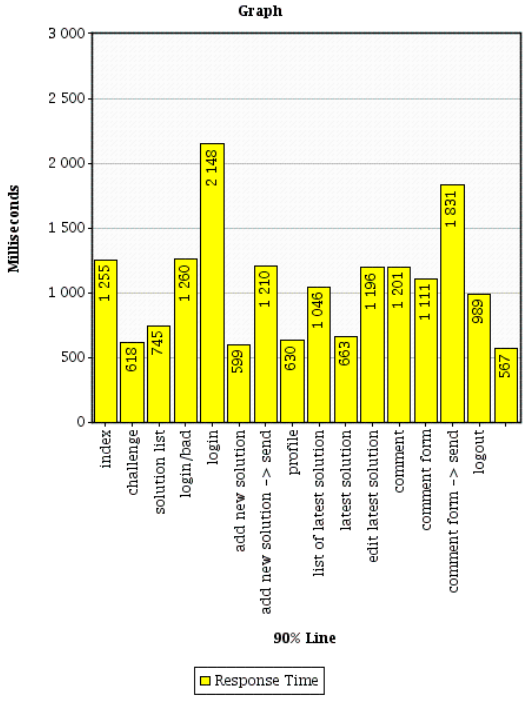
Příloha C: Výsledky testování Apache JMeter



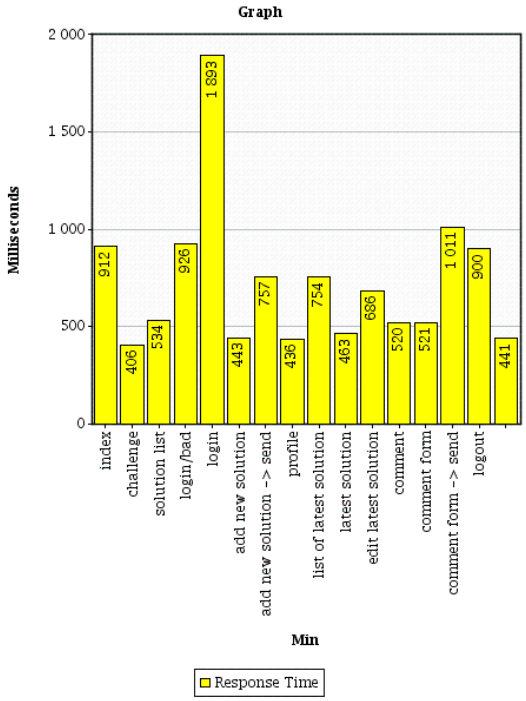
Test číslo 1: Průměrná odezva



Test číslo 1: Maximální odezva

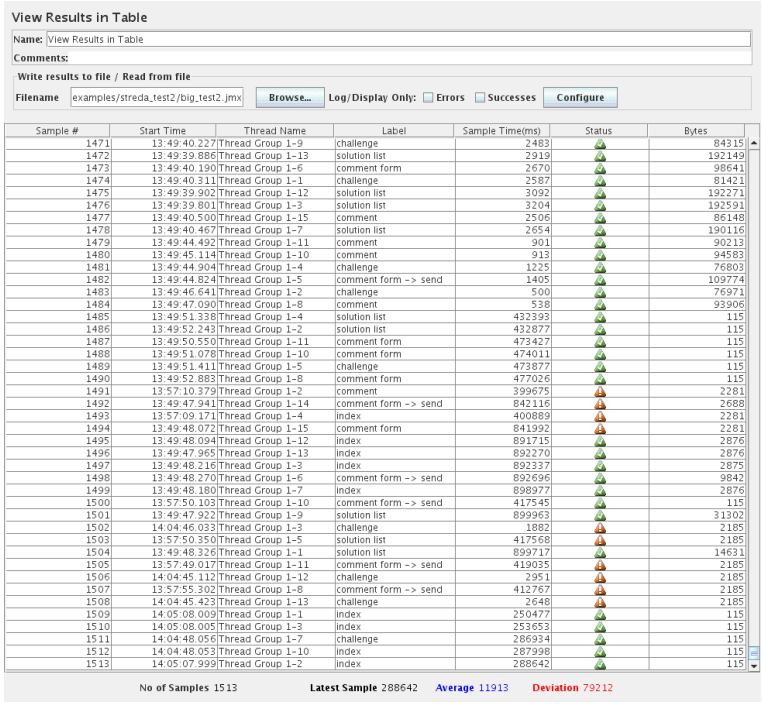


Test číslo 1: 90% precentil odezva

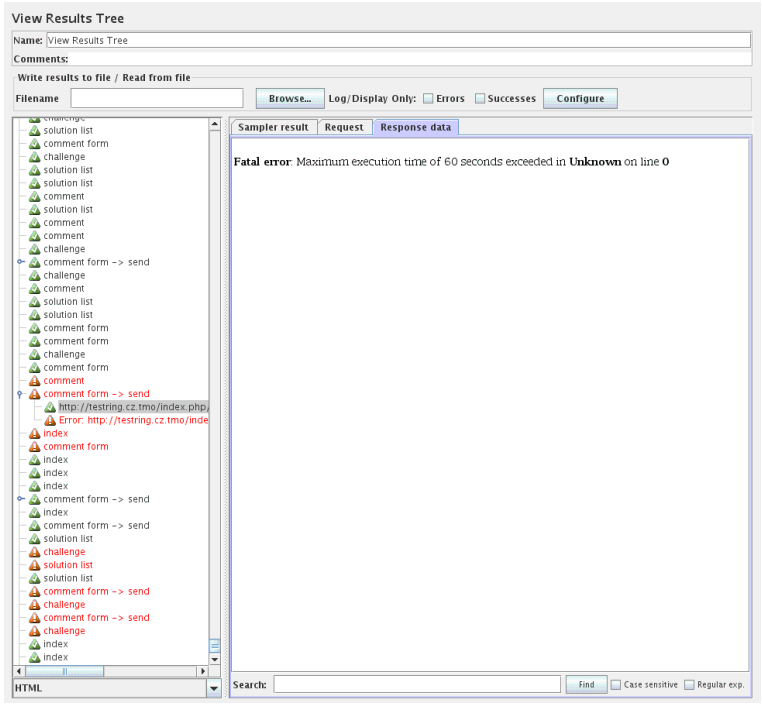


Test číslo 1: Minimální odezva

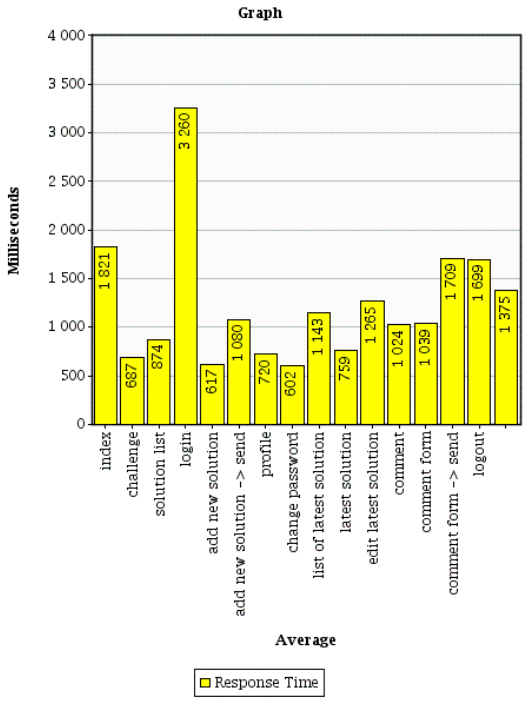
Test číslo 2: Průběh testu



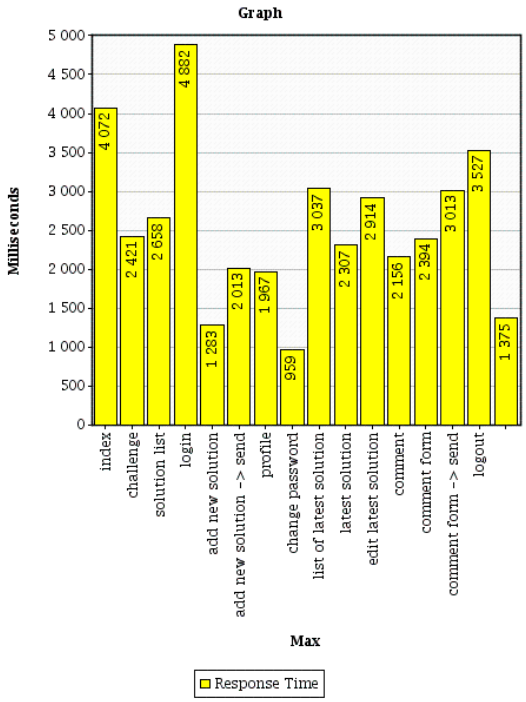
Test číslo 2: Tabulka s výsledky, je vidět několik chybových stavů



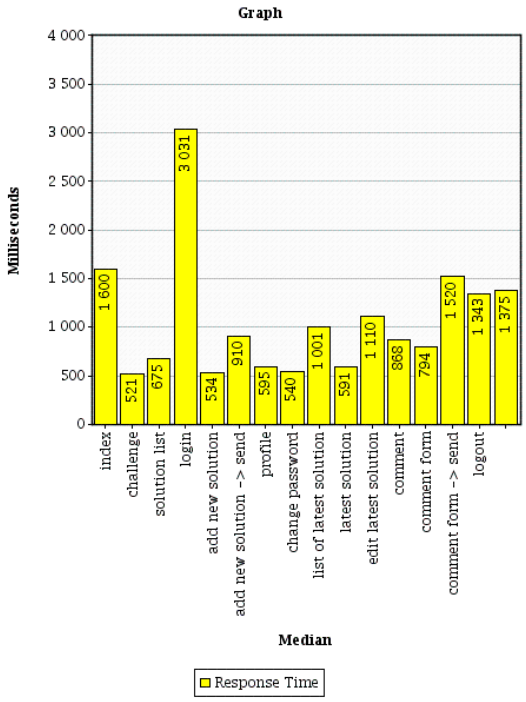
Test číslo 2: Chyba, překročen časový limit



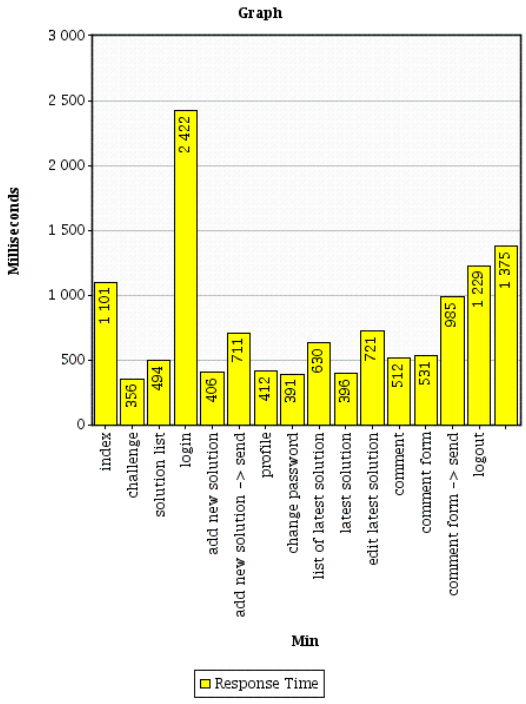
Test číslo 3: Průměrná odezva



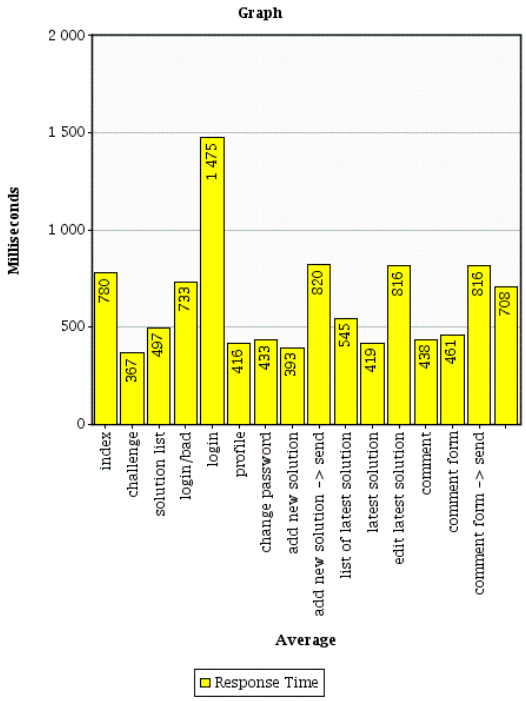
Test číslo 3: Maximální odezva



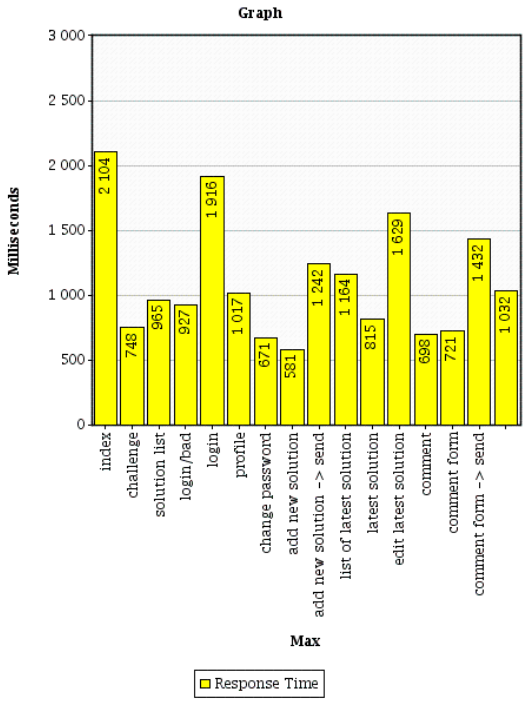
Test číslo 3: 90% percentil odezva



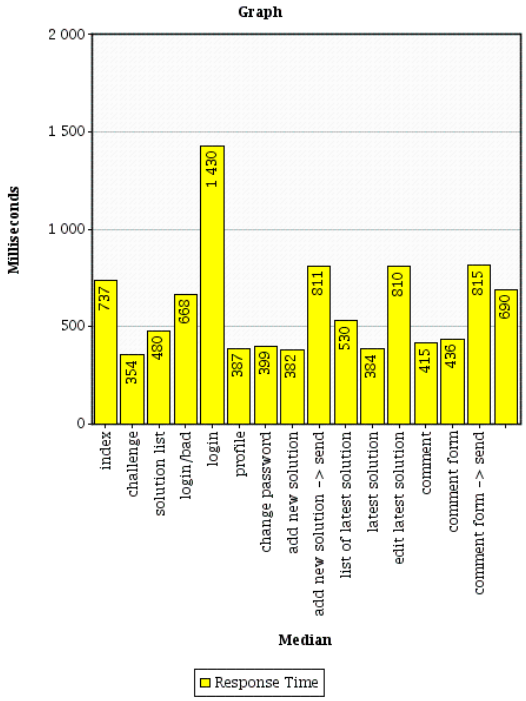
Test číslo 3: Minimální odezva



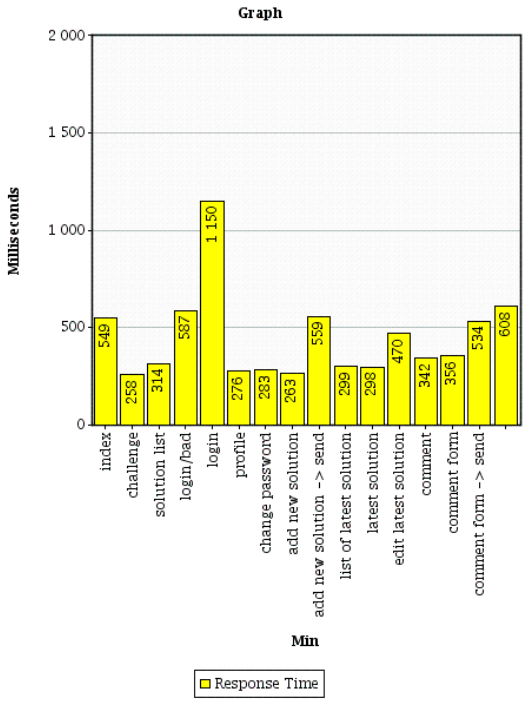
Test číslo 4: Průměrná odezva



Test číslo 4: Maximální odezva



Test číslo 4: 90% precentil odezva



Test číslo 4: Minimální odezva

Příloha D: Testovací scénář z nástroje The Grinder

```
from net.grinder.plugin.http import HTTPRequest, HTTPPluginControl,
HTTPPluginConnection
from HTTPClient import Cookie, CookieModule, CookiePolicyHandler
from java.util import Date

from net.grinder.script.Grinder import grinder
from net.grinder.script import Test
import java.lang.String as String

from HTTPClient import NVPair

#head
from net.grinder.plugin.http import HTTPPluginControl
HTTPPluginControl.getConnectionDefaults().followRedirects = 1

cas = 1000
url = "http://testring.cz.tmo/"

class MyCookiePolicyHandler(CookiePolicyHandler):
      def acceptCookie(self, cookie, request, response):
              #output("Accept cookie: %s" % cookie)
              return 1

      def sendCookie(self, cookie, request):
              #output("Send cookie: %s" % cookie)
              return 1

CookieModule.setCookiePolicyHandler(MyCookiePolicyHandler())

test1 = Test(1, "users: 6, intervals: 5s")
request = test1.wrap(HTTPRequest())

output = grinder.logger.output


#main---------------------------------------------------------------------
-----------------------------------
class TestRunner:
      def __call__(self):

            thread_id = grinder.threadNumber

            #-----select user by thread
            if thread_id == 0:
                  user = "lukas.vosahlik"
            elif thread_id == 1:
                  user = "ondrej.choc"
            elif thread_id == 2:
                  user = "milos.kalhous"
```

```
            elif thread_id == 3:
                  user = "vit.dolejsi"
            elif thread_id == 4:
                  user = "roland"
            elif thread_id == 5:
                  user = "barbora"
            elif thread_id == 6:
                  user = "frank"
            elif thread_id == 7:
                  user = "otakar"
            elif thread_id == 8:
                  user = "petr.fabian"
            elif thread_id == 9:
                  user = "romana.blazova"
            elif thread_id == 10:
                  user = "milan.majercik"
            elif thread_id == 11:
                  user = "vojtech.sazel"
            elif thread_id == 12:
                  user = "jan.smocek"
            elif thread_id == 13:
                  user = "ales.adensam"
            elif thread_id == 14:
                  user = "ondrej.cibula"

            #-----start
            output("\n\n-----start as: \"" + user + "\", thead: " +
str(thread_id) + "-----\n")

            #-----index
            output("-----index-----")
            response = request.GET(url)
            grinder.sleep(cas)

            #-----browsing/anonymous in cycle
            for a in range(0, (int(random_gen()) + 1) ):
                  browsing()

            #-----login/bad
            if int(random_gen()) < 3:
                  login_bad(user)

            #-----login
            login(user)

            #-----change password
            if int(random_gen()) < 3:
                  change_password()

            #-----add solution
            add_solution()

            #-----edit solution
            for a in range(0, 10):
```

```
                edit_solution()

            #-----add comment
            for a in range(0, 3):
                add_comment()

            #-----browsing/logged
            for a in range(0, (int(random_gen()) + 1) ):
                browsing()

            #-----logout
            logout()

            #-----end
            output("\n\n-----end-----\n")


#---------------------------------------------------------------------------
-----------------------------------
def random_gen():
      current_date = Date()
      sec = current_date.getSeconds()
      string = str(sec)

      if len(string) == 1:
            return string[0]
      else:
            return string[1]


def extract_token(token, find_token, lenght, lenght2, sign):

      do = token.find(find_token)

      if sign == 0:
            od = do + lenght
            do = do + lenght2
      else:
            do = do - lenght
            od = do - lenght2

      token2 = token[od:do]

      return token2


def browsing():

      output("\n\n-----browsing-----\n")

      #-----index
      output("-----index-----")
      request.GET(url)
      grinder.sleep(cas)
```

```
	#-----challenge
	random_gen1 = random_gen()
	challenge = "35"
	if random_gen1 == ("0" or "5"):
		challenge = "37"
	elif random_gen1 == ("2" or "7"):
		challenge = "29"
	elif random_gen1 == ("4" or "1"):
		challenge = "36"
	elif random_gen1 == ("6" or "3"):
		challenge = "15"
	elif random_gen1 == ("8"):
		challenge = "28"
	elif random_gen1 == ("9"):
		challenge = "35"
	output("-----challenge-----")
	request.GET(url + "index.php/challenge/detail/id_chl/" + challenge)
	grinder.sleep(cas)

	#-----solution
	output("-----solution-----")
	request.GET(url + "index.php/solution/solutionList/id_chl/" +
challenge)
	grinder.sleep(cas)

	return


def login(user_name):

	output("\n\n-----login-----\n")

	#-----index
	output("-----index-----")
	response = request.GET(url)
	grinder.sleep(cas)

	YII_CSRF_TOKEN = extract_token( str( response.getText() ),
"YII_CSRF_TOKEN", 8, 40, 1)
	output("-----extract----- Extracted token(YII_CSRF_TOKEN) is: " +
YII_CSRF_TOKEN)

	#-----cookie
	threadContext = HTTPPluginControl.getThreadHTTPClientContext()

	expiryDate = Date()
	expiryDate.year += 1
	cookie = Cookie("YII_CSRF_TOKEN", YII_CSRF_TOKEN, url, "/",
expiryDate, 0)
	#CookieModule.addCookie(cookie, threadContext)

	#-----login
	login_data = [NVPair("YII_CSRF_TOKEN", YII_CSRF_TOKEN),
```

```
NVPair("LoginForm[username]", user_name), NVPair("LoginForm[password]",
"test")]
      output("-----login----- as " + user_name)
      response = request.POST(url + "index.php", login_data)
      grinder.sleep(cas)

      #>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>.
      thread_id2 = grinder.threadNumber
      pomocna = response.getText()
      if pomocna.find("odhlásit") != -1:
            output("\n\nThread: " + str(thread_id2) + " se uspesne
prihlasil.\n")
      else:
            output("\n\nThread: " + str(thread_id2) + " se
neprihlasil.\n")
      #output( response.getText() )

      return


def login_bad(user_name):

      output("\n\n-----login/bad-----\n")

      #-----index
      output("-----index-----")
      response = request.GET(url)
      grinder.sleep(cas)

      YII_CSRF_TOKEN = extract_token( str( response.getText() ),
"YII_CSRF_TOKEN", 8, 40, 1)
      output("-----extract----- Extracted token(YII_CSRF_TOKEN) is: " +
YII_CSRF_TOKEN)

      #-----cookie
      threadContext = HTTPPluginControl.getThreadHTTPClientContext()

      expiryDate = Date()
      expiryDate.year += 1
      cookie = Cookie("YII_CSRF_TOKEN", YII_CSRF_TOKEN, url, "/",
expiryDate, 0)
      #CookieModule.addCookie(cookie, threadContext)

      #-----login
      login_data = [NVPair("YII_CSRF_TOKEN", YII_CSRF_TOKEN),
NVPair("LoginForm[username]", user_name), NVPair("LoginForm[password]",
"test2")]
      output("-----login----- as " + user_name)
      response = request.POST(url + "index.php", login_data)
      grinder.sleep(cas)
      #output( response.getText() )

      return
```

```
def change_password():

      output("\n\n-----change password-----\n")

      #-----index
      output("-----index-----")
      response = request.GET(url)
      grinder.sleep(cas)

      #-----profile
      link = extract_token( str( response.getText() ),
"mainmenu_useremail", 27, 63, 0)
      output("-----extract----- Extracted token(link) is: " + link)
      output("-----profile-----")
      response = request.GET(url + link)
      grinder.sleep(cas)

      #-----change password
      link = extract_token( str( response.getText() ), 'ButtonText_9"',
21, 60, 0)
      output("-----extract----- Extracted token(link) is: " + link)
      YII_CSRF_TOKEN = extract_token( str( response.getText() ),
"YII_CSRF_TOKEN", 8, 40, 1)
      output("-----extract----- Extracted token(YII_CSRF_TOKEN) is: " +
YII_CSRF_TOKEN)
      new_password = [NVPair("YII_CSRF_TOKEN", YII_CSRF_TOKEN),
NVPair("UpdatePasswordForm[currentPassword]", "test"),
NVPair("UpdatePasswordForm[newPassword]", "test"),
NVPair("UpdatePasswordForm[newPasswordVerify]", "test")]
      output("-----change password-----")
      response = request.POST(url + link, new_password)
      #output( response.getText() )
      grinder.sleep(cas)

      return

def add_solution():

      output("\n\n-----add solution-----\n")

      #-----index
      output("-----index-----")
      request.GET(url)
      grinder.sleep(cas)

      #-----challenge
      random_gen1 = random_gen()
      challenge = "35"
      if random_gen1 == ("0" or "5"):
            challenge = "37"
      elif random_gen1 == ("2" or "7"):
            challenge = "29"
      elif random_gen1 == ("4" or "1"):
```

```
            challenge = "36"
      elif random_gen1 == ("6" or "3"):
            challenge = "15"
      elif random_gen1 == ("8"):
            challenge = "28"
      elif random_gen1 == ("9"):
            challenge = "35"
      output("-----challenge-----")
      request.GET(url + "index.php/challenge/detail/id_chl/" + challenge)
      grinder.sleep(cas)

      #-----list of solutions
      output("-----solution-----")
      response = request.GET(url +
"index.php/solution/solutionList/id_chl/" + challenge)
      grinder.sleep(cas)

      #-----add new solution
      link = extract_token( str( response.getText() ), 'class="
textbutton forth"', 52, 104, 0)
      output("-----extract----- Extracted token(link) is: " + link)
      output("-----add new solution-----")
      request.GET(url + link)
      grinder.sleep(cas)

      #-----add new solution -> send
      YII_CSRF_TOKEN = extract_token( str( response.getText() ),
"YII_CSRF_TOKEN", 8, 40, 1)
      output("-----extract----- Extracted token(YII_CSRF_TOKEN) is: " +
YII_CSRF_TOKEN)
      output("-----add new solution -> send-----")
      current_date = Date()
      current_time = str(current_date.getTime())
      solution_data = [NVPair("YII_CSRF_TOKEN", YII_CSRF_TOKEN),
NVPair("Solution[text_short_sol]", "Zapsano - test The Grinder 3 " +
current_time)]
      response = request.POST(url + link, solution_data)
      #output( response.getText() )
      grinder.sleep(cas)

      return


def edit_solution():

      output("\n\n-----edit solution-----\n")

      #-----index
      output("-----index-----")
      response = request.GET(url)
      grinder.sleep(cas)

      #-----profile
      link = extract_token( str( response.getText() ),
```

```
"mainmenu_useremail", 27, 63, 0)
      output("-----extract----- Extracted token(link) is: " + link)
      output("-----profile-----")
      response = request.GET(url + link)
      grinder.sleep(cas)

      #-----list of latest solutions
      link = extract_token( str( response.getText() ),
"showProfile_solutinsCount", 34, 75, 0)
      output("-----extract----- Extracted token(link) is: " + link)
      output("-----list of latest solutions-----")
      response = request.GET(url + link)
      grinder.sleep(cas)

      #-----latest solution
      link = extract_token( str( response.getText() ), "Menu_12_1", 18,
71, 0)
      output("-----extract----- Extracted token(link) is: " + link)
      output("-----latest solution-----")
      response = request.GET(url + link)
      grinder.sleep(cas)

      #-----edit latest solution
      YII_CSRF_TOKEN = extract_token( str( response.getText() ),
"YII_CSRF_TOKEN", 8, 40, 1)
      output("-----extract----- Extracted token(YII_CSRF_TOKEN) is: " +
YII_CSRF_TOKEN)
      current_date = Date()
      current_time = str(current_date.getTime())
      solution_data = [NVPair("YII_CSRF_TOKEN", YII_CSRF_TOKEN),
NVPair("Solution[text_short_sol]", "Editovano - test The Grinder 3 " +
current_time)]
      output("-----edit latest solution-----")
      response = request.POST(url + link, solution_data)
      grinder.sleep(cas)

      return


def add_comment():

      output("\n\n-----add comment-----\n")

      #-----index
      output("-----index-----")
      request.GET(url)
      grinder.sleep(cas)

      #-----challenge
      random_gen1 = random_gen()
      challenge = "35"
      if random_gen1 == ("0" or "5"):
            challenge = "37"
      elif random_gen1 == ("2" or "7"):
```

```
            challenge = "29"
      elif random_gen1 == ("4" or "1"):
            challenge = "36"
      elif random_gen1 == ("6" or "3"):
            challenge = "15"
      elif random_gen1 == ("8"):
            challenge = "28"
      elif random_gen1 == ("9"):
            challenge = "35"
      output("-----challenge-----")
      request.GET(url + "index.php/challenge/detail/id_chl/" + challenge)
      grinder.sleep(cas)

      #-----list of solutions
      output("-----solution-----")
      response = request.GET(url +
"index.php/solution/solutionList/id_chl/" + challenge)
      grinder.sleep(cas)

      #-----comment
      output("-----comment-----")
      current_date = Date()
      sec = current_date.getSeconds()
      id_sol = str(sec + 50)
      response = request.GET(url + "index.php/comment/emptyList/id_sol/"
+ id_sol)
      grinder.sleep(cas)

      #-----comment form
      output("-----comment form-----")
      response = request.GET(url +
"index.php/comment/commentForm/id_sol/" + id_sol)
      grinder.sleep(cas)

      #-----comment form -> send
      YII_CSRF_TOKEN = extract_token( str( response.getText() ),
"YII_CSRF_TOKEN", 8, 40, 1)
      output("-----extract----- Extracted token(YII_CSRF_TOKEN) is: " +
YII_CSRF_TOKEN)
      current_date = Date()
      current_time = str(current_date.getTime())
      comment_data = [NVPair("YII_CSRF_TOKEN", YII_CSRF_TOKEN),
NVPair("Comment[text_short_com]", "Comment - test The Grinder 3 " +
current_time)]
      output("-----comment form -> send-----")
      response = request.POST(url +
"index.php/comment/commentForm/id_sol/" + id_sol, comment_data)
      grinder.sleep(cas)
      #output( response.getText() )

      return


def logout():
```

```
output("\n\n-----logout-----\n")

#-----index
output("-----index-----")
request.GET(url + "index.php/site/logout")
grinder.sleep(cas)

return
```